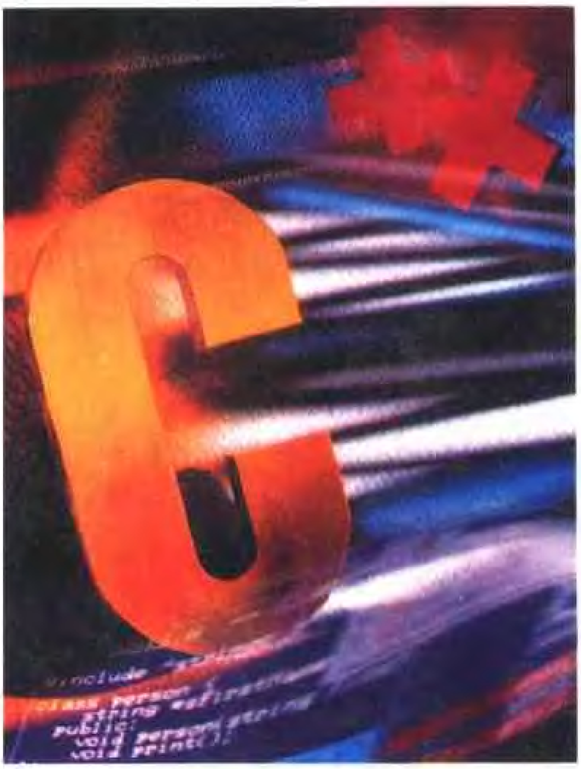
TURBO C

绘图程序设计实例

庄粤盛
何敏煌　编著



学苑出版社

希望漢字系統

UCDOS Ver 3.0

▲ 支持直接写屏，中西文兼容，英文制表符识别效果最佳，如实再现原版软件的神奇风采

▲ 充分利用扩展內存，可实现零字节占用，不受DOS版本限制

▲ 配备WPS文字处理系统， 可在网络环境下正常运行，并可同时使用26种矢量字库进行打印

▲ 眞正实现网络共享，工作站数目不受限制，彻底解决共享打印和远程通讯

▲ 矢量汉字打印速度奇快，无与伦比

▲ 可在各种图形模式下显示任意大小的文字，并具有屏幕作图、背景音乐和图像保存恢复功能

▲ 提供多种汉字输入方法(含五笔)，记忆词组方便灵活，可在多种输入法中随意使用

單用戶:￥880.00元　網絡版:￥2000.00元　老用戶升級優惠

地　址：北京市海淀路82号 8721信箱　聯系人：周东 夏克 陈江
電　話：01-2541992,2562329　傳　真：01-2561057
開　户：北京海淀工商行中关村城市信用社
户　名：北京希望电脑公司　帳　號：05079-08

北京希望电脑公司计算机程序设计语言系列丛书

松崗
XWC038

ISBN 7—5077—0807—1/TP. 18

本册定价：49.00 元(含盘)

计算机程序设计语言系列丛书

Turbo C 绘图程序设计实例

庄粤盛　何敏煌　编著
天　奥　改编
熊可宜　审校

学 苑 出 版 社

1993

（京）新登字 151 号

内 容 摘 要

本书介绍了如何使用 Turbo C 语言进行绘图程序设计的技巧和方法，内容包括图形显示技巧、图形文件格式、Super VGA 绘图程序设计、图形输出程序，常驻内存程序的设计以及扫描器应用的程序设计，除了基础原理阐述之外，并附有多种范例以供读者参考。

欲购本书的用户，请直接与北京 8721 信箱联系，电话：2562329，邮政编码：100080。

版 权 声 明

本书繁体字中文版原书名为《Turbo C 绘图程式设计实务》，由松岗电脑图书资料股份有限公司出版，版权归松岗公司所有。本书简体字的中文版版权由松岗公司授予北京希望电脑公司，由北京希望电脑公司和学苑出版社独家出版、发行，未经出版者书面许可，本书的任何部分不得以任何手段复制或传播。

计算机程序设计语言系列丛书

Turbo C 绘图程序设计实例

编　　著：庄粤盛　何敏煌
改　　编：天　奥
审　　校：熊可宜
责任编辑：徐建军
出版发行：学苑出版社　　邮政编码：100032
社　　址：北京市西城区成方街 33 号
排　　版：天奥科技公司照排中心排版
印　　刷：双青印刷厂
开　　本：787×1092　1/16
印　　张：22.3　　字数：520 千字
印　　数：1—5000 册
版　　次：1993 年 10 月北京第 1 版第 1 次
ISBN 7—5077—0807—1/TP・18
本册定价：49.00 元（含盘）

学苑版图书印、装错误可随时退换

序 言

本书是针对稍具C语言基础的读者而编写的。在书中，我们以计算机绘图为主题，由浅入深、循序渐进地介绍几个绘图程序应用实例，供读者参考。

计算机绘图是相当吸引人的研究主题，目前在大多数的应用软件中，或多或少都会用到绘图技巧，尤其对于青少年学生而言，色彩鲜明、生动活泼的图形，远比财务管理、数据库、文字处理等程序更具吸引力。笔者深深觉得：市面上讨论绘图的书籍很多，然而往往仅限于基础原理说明及绘图函数的使用介绍；否则便是过于专业，非一般读者所能接受。以本书各章的主题而言，已经有些先后在相关杂志、期刊上介绍过了。然而或许限于篇幅，这些文章并未做深入探讨，以致于阅读之后，虽然极感兴趣，可惜资料有限，无法探究更深一层技巧，或做更广泛的应用。

笔者有鉴于此，经过研究相关书籍、资料，经实际编写程序加以印证后，编写成书，愿与读者相互交换心得。本书所探讨的主题有下列几点：

1. 图形显示技巧
2. 图形文件格式
3. Super VGA 绘图程序设计
4. 绘图与打印图形程序
5. 以C语言编写常驻程序
6. 扫描仪应用程序

在书中的每一章，除了基本原理阐述之外，并开发多种范例程序附于各章之中，以供读者参考。我们深深希望各位阅读之后，会有所助益。

目　录

第一章　概论 …… 1
1.1　前言 …… 1
1.2　屏幕与显示适配器 …… 1
1.3　常见的绘图软件 …… 6
1.4　Turbo C 的绘图函数 …… 6
第二章　图形显示技巧 …… 13
2.1　概述 …… 13
2.2　图形显示基础知识 …… 13
2.3　各种图形显示方法 …… 20
2.4　实例汇聚 …… 54
2.5　结论 …… 64
第三章　图形文件格式 …… 65
3.1　简介 …… 65
3.2　各种图形文件的格式 …… 67
3.3　文件格式的转换 …… 98
3.4　图形文件的解读技巧 …… 129
第四章　彩色图形文件的读取 …… 135
4.1　彩色图形的储存方式 …… 135
4.2　Turbo C 的 getimage 彩色图形储存格式 …… 137
4.3　彩色 PUT 图形文件的读取 …… 141
4.4　彩色 MAP 图形文件的读取 …… 144
4.5　彩色 GPM 图形文件的读取 …… 145
4.6　彩色 TPC 图形文件的读取 …… 148
4.7　彩色 BPC 图形文件的读取 …… 149
4.8　彩色 PCX 图形文件的读取 …… 151
第五章　Super VGA 绘图程序设计 …… 154
5.1　Super VGA 简介 …… 154
5.2　Super VGA 的结构 …… 156
5.3　使用 BIOS 常用程序绘图 …… 158
5.4　直接存取 Super VGA 显示内存 …… 174
5.5　硬件图形放大功能 …… 191
5.6　结束语 …… 209
第六章　绘图与打印图形程序设计 …… 210
6.1　图形打印的基本原理 …… 210

6.2 单色打印图形程序设计 …… 216
6.3 彩色图形打印 …… 221
6.4 绘图程序设计 …… 224
第七章 驻留式绘图程序设计实例 …… 253
7.1 驻留程序简介 …… 253
7.2 Turbo C 编写驻留程序的方法 …… 253
7.3 简单的驻留程序示例 …… 257
7.4 加强驻留程序的功能 …… 259
7.5 驻留程序的实例 …… 267
7.6 单色驻留打印图形程序 …… 296
7.7 彩色驻留式打印图形程序 …… 306
7.8 驻留绘图程序设计——绘图小精灵 …… 313
7.9 结束语 …… 341
第八章 扫描仪应用程序 …… 342
8.1 扫描仪简介 …… 342
8.2 扫描仪应用程序设计实例 …… 342
8.3 结束语 …… 351

第一章 概论

1.1 前 言

计算机绘图是目前相当热门的研究主题，在一般的计算机展览会中，最能吸引观众目光的往往就是计算机绘图了。无论在计算机辅助设计、美工设计、计算机游戏、简报系统和计算机辅助教学等软件中，都可以看见它以生动活泼的画面，吸引观众的注意力。

由于计算机绘图需要大量的内存及较快的执行速度，几年之前，它可说是大型计算机的专利品，在微电脑上并不普遍。一般人要编写绘图程序时往往需要自行以汇编语言开发相关的程序库和驱动软件；这些工作并非初学者所能胜任的。随着计算机科技的进步，计算机绘图也逐渐走入生活中，如今在微计算机上已经能够执行功能强大的绘图程序了。例如，飞行模拟，3D 动画图像处理等复杂的程序，过去只能在工作站级以上的计算机上执行，而目前 80386，80486CPU 的微电脑执行这些程序，已经绰绰有余了。在开发工具方面，许多高级语言都已经提供了若干绘图函数，使得绘制图形比以前更为方便。例如 Borland 公司所开发的 Turbo C 在 1.5 版提供了许多功能强大的绘图函数，而在 2.0 版又新增了若干绘图函数，使得 C 语言的绘图功能更加完备。请记着，我们即使不懂汇编语言，也能着手编写绘图程序，共享计算机绘图的乐趣。

本书是针对稍具程序设计基础的读者而编写的。我们以专题制作的方式，探讨在绘图程序设计时，经常会接触到的几个主题。书中所有的程序皆以 Turbo C 编写，因为它简单易学，同时又提供了完备的绘图程序库。在本章中，我们先对本书相关的软件、硬件环境做一个简单的介绍。

1.2 屏幕与显示适配器

计算机绘图的基本设备除了主机之外，还有屏幕和显示适配器，三者的关系如图 1−1 所示。

计算机的显示器（Monitor）俗称为屏幕，它的功能是将计算机所处理的文字、图形显示出来，而显示的色彩和分辨率，则与显示适配器（Video Adapter）有直接的关系。

微电脑常用的屏幕有数字式和模拟式两种：

1. 数字式（Digital TTL Displays）：

输入信号线有若干条，每条以 ON／OFF 两种状态表示；如果有 N 条输入线，则屏幕可以产生 2^N 种色彩，EGA 即采用这种方式。

2. 模拟式（Analog RGB Displays）：

以三条输入信号线分别代表红、绿、蓝三种色彩，信号的强度则代表该颜色的深浅，由这三种颜色可组合成任意色彩。目前常用的 VGA 和 Super VGA 就是采用此种方式。

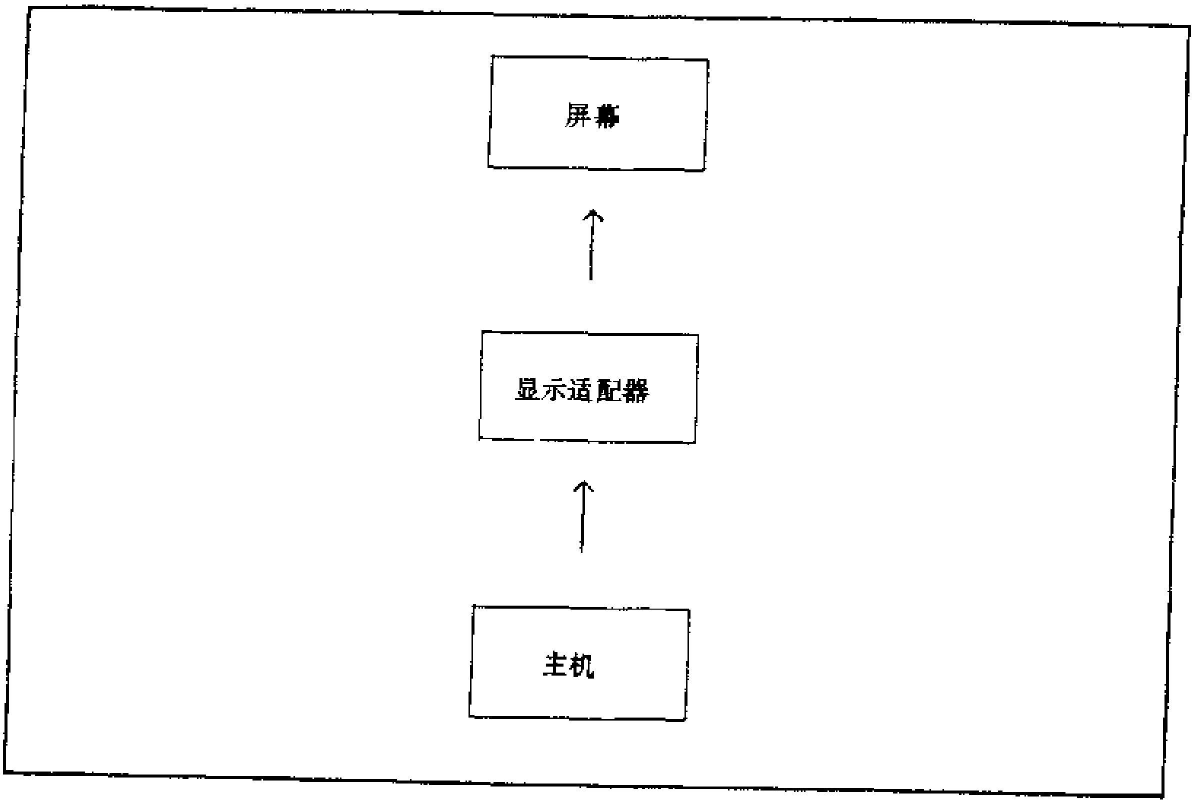


图 1-1 计算机绘图的基本设备

另外有一种混合式（Composite）屏幕，只有一条输入信号线，而以解码的方式输出信号．一般用于电视上，在微电脑上较少采用，CGA 即是混合式的屏幕．

我们若要将图形显示于屏幕上，必须将图形数据预先存入与屏幕相对应的内存中．这块与屏幕相对应的内存称为显示内存，位于显示适配器上，它的大小则视适配器的种类而定．当我们要显示图形时，只要图形的每一个点放到显示内存的相对应的地址，就可以将图形呈现在屏幕上，这种方式称为内存映射式（Memory Mapped）．

以下所介绍的是微电脑常用的显示适配器．

1.2.1 单色显示适配器

早期 IBM PC 所配备的显示适配器是单色的 Monochrome Display Adapter 简称 MDA，它的分辨率为 720×350，但只能显示文字，无法处理图形，现在已经很少见了．

Hercules 卡是美国 Hercules 公司所出品的单色绘图卡，它提供了文字和绘图两种显示模式，分辨率为 720×348，不仅具备了 MDA 卡的所有功能，并且分辨率也比 CGA 卡高．Hercules 卡在国内很受欢迎，目前很多单色屏幕都配备此种适配器．

1.2.2 CGA 卡

CGA 卡（Color Graphics Apdpter）是 IBM PC 最早使用的彩色适配器．由于分辨率仅 640×200，又只有四色，并且无法显示中文，因此这种显示适配器在国内很少见。

1.2.3 EGA 卡

IBM 公司在 1984 年推出 EGA 卡（Enhanced Graphics Adapter）。它包含了 CGA

卡的所有功能，并且增加了 640×200 16 色、640×350 单色和 640×350 16 色等绘图模式，功能远胜于 CGA 卡，目前较新的绘图软件大多支持这种适配器。

1.2.4 VGA 卡

IBM 公司于 1987 年配合 PS／2 系列微电脑推出了 VGA 卡（Video Graphics Array），造成微电脑屏幕显示系统的大震撼。VGA 卡不仅包含了所有的 EGA 显示模式，更将显示能力提升到 640×480 16 色。在此之前，屏幕的输出是数字信号，而 VGA 卡则以模拟信号输出，使得屏幕能够表现的色彩更为丰富。VGA 卡上有 256 个颜色暂存器，因此它最多能在屏幕上同时显示 256 色。标准 IBM VGA 所具有的显示模式如表 1-1 所示。在 VGA 卡推出之后，许多软件公司纷纷以它为标准，开发应用软件：而有些硬件厂商也以此标准制造各种 VGA 卡，因此我们称 IBM VGA 卡为现阶段的彩色标准适配器。

表 1-1 标准 IBM VGA 所具有的绘图模式

模式编号（16 进制）	形态	分辨率	颜色	缓冲区段址	MDA	CGA	EGA	MCGA	VGA
0	文本	40×25 字 (320×200)	16	B800		✓	✓	✓	✓
0	文本	40×25 字 (320×350)	16	B800			✓		✓
0	文本	40×25 字 (320×400)	16	B800				✓	
0	文本	40×25 字 (360×400)	16	B800					✓
1	文本	40×25 字 (320×200)	16	B800		✓	✓	✓	✓
1	文本	40×25 字 (320×350)	16	B800			✓		✓
1	文本	40×25 字 (320×400)	16	B800				✓	
1	文本	40×25 字 (360×400)	16	B800					✓
2	文本	80×25 字 (640×200)	16	B800		✓	✓	✓	✓
2	文本	80×25 字 (640×350)	16	B800			✓		✓
2	文本	80×25 字 (640×400)	16	B800				✓	
2	文本	80×25 字 (720×400)	16	B800					✓

(续)

模式编号(16进制)	形态	分辨率	颜色	缓冲区段址	MDA	CGA	EGA	MCGA	VGA
3	文本	80×25字 (640×200)	16	B800		✓	✓	✓	✓
3	文本	80×25字 (640×350)	16	B800			✓		✓
3	文本	80×25字 (640×400)	16	B800				✓	
3	文本	80×25字 (720×400)	16	B800					✓
4	图形	320×200	4	B800		✓	✓	✓	✓
5	图形	320×200	4	B800		✓	✓	✓	✓
6	图形	640×200	2	B800		✓	✓	✓	✓
7	文本	80×25字 (720×350)	2	B800	✓		✓		✓
7	文本	80×33字 (720×400)	2	B800					✓
8	仅Pcjr								
9	仅Pcjr								
0A	仅Pcjr								
0B	(EGA Video BIOS 使用)								
0C	(EGA Video BIOS 使用)								
0D	图形	320×200	16	A000			✓		✓
0E	图形	640×200	16	A000			✓		✓
0F	图形	640×350	2	A000			✓		✓
10	图形	640×350	4	A000			✓		✓
10	图形	640×350	16	A000			✓		✓
11	图形	640×480	2	A000				✓	✓
12	图形	640×480	16	A000					✓
13	图形	320×200	256	A000				✓	✓

1.2.5 Super VGA 图形适配器

目前市面上可以看见几种比 IBM VGA 卡功能更强的彩色显示适配器，它们的分辨率比 VGA 更高，并且提供更多的颜色，我们称这些显示适配器新增加的功能为 Super VGA 扩充图形模式。

Super VGA 若使用 256K 的显示内存，能提供 800×600 16 色或者 640×350 256 色的彩色绘图能力。如果采用 512K 的显示内存时，它的绘图能力则达到 640×480 256

色、800×600 256色以及1024×768 16色。

具有Super VGA扩充图形模式的适配器很多，例如ATI，Tseng，Triden等等。由于Super VGA尚未标准化，每个厂牌的规格并不统一，在选用时，必须先行阅读技术手册，才能应用自如。表1-2是Tseng ET3000（曾氏卡）的Super VGA扩充绘图模式。

表1-2 ET3000卡新增的显示模式

模式编号(16进制)	形态	分辨率	颜色	内存需求	显示形态
18	文本	132×44	单色	256KB	Super VGA
19	文本	132×25	单色	256KB	Super VGA
1A	文本	132×28	单色	256KB	Super VGA
22	文本	132×44	16	256KB	Super VGA
23	文本	132×25	16	256KB	Super VGA
24	文本	132×28	16	256KB	Super VGA
25	图形	640×480	16	256KB	VGA
26	文本	80×60	16	256KB	VGA
27	图形	720×512	16	256KB	Super VGA
29	图形	800×600	16	256KB	Super VGA
2D	图形	640×350	256	512KB	VGA
2E	图形	640×480	256	512KB	VGA
2F	图形	720×512	256	512KB	Super VGA
30	图形	800×600	256	512KB	Super VGA
37	图形	1024×768	16	512KB	8541.XL

由于Super VGA尚未标准化，因此一般的程序语言都没有提供扩充模式的绘图函数，Turbo C就是这种例子。本书针对此点，特别在第五章的各节中，和读者详细介绍其基本构造及程序设计的要领。

1.2.6 XGA卡

XGA(Extended Graphics Array)是IBM公司最新推出的显示适配器。在此之前，该公司所推出的CGA，EGA，VGA等适配器都曾经成为当时的标准，而目前这些适配器已经无法满足用户的需求了。IBM公司虽然曾推出8514／A显示适配器，它的绘图能力最高可达1024×768 16色，然而它与VGA卡并不相容，因此无法成为标准。有鉴于此，IBM公司更积极开发了XGA卡，期望它能继VGA卡后，再度成为PC彩色绘图适配器的标准。

XGA包含了三种模式：

1. VGA相容模式：

XGA不仅与VGA完全相容，其显示速度也比VGA快了许多。IBM宣称：XGA在DOS下比VGA卡快90%，在Windows下比VGA快31%，比Super VGA快50%。

2. 132 行文本模式:

在 XGA 下，每个字符的宽度为 8 个像素，在 1056 的水平分辨率下，可以显示 132 行文字。由于字符的高度可依字体而定，因此我们可以选择 132×43，132×50 或者 132×60 等文本显示模式。

3. 扩展图形(Extended Graphics)模式:

XGA 最引人注目的就是它的扩展图形功能，例如，16 位的真实色彩能力、分辨率可达 640×480 65536 色或者 1024×768 256 色，其它聚集像素(Packed Pixel)处理、硬件光标等，都是目前 Super VGA 所不及的。

1.3 常见的绘图软件

在 PC 上使用的绘图软件很多，其中国内读者最为熟悉的就是 AutoCAD 和 PC PaintBrush 两套软件了。

PC Paint Brush 是 Zsoft 公司在 1985 年推出的位式绘图软件，它提供了多种绘图工具，在进行绘图、填色、喷色、输入文字时都非常方便。此外还可以将图形做旋转、放大、缩小、反相、倾斜等变化，图形绘制和编辑的功能十分完备。在 Windows 3.0 的辅助软件中，即包含了 PaintBrush 应用程序。

AutoCAD 是 AutoDesk 公司的产品。它是计算机辅助设计最具有代表性的软件，一般性的工程图如机械设计图、建筑设计图等，都少不了它。

除了上述两种之外，Animator，TOPAS，Grasp，OrCAD，Rambow Paint 和达文西、变影 72、百图等等，都是国内常见的绘图软件。

1.4 Turbo C 的绘图函数

Turbo C 在 1.5 版中加人了绘图函数，而在 2.0 版时，更将绘图函数增加至 100 个左右。这些函数存在 GRAPHICS.LIB 库中，而它们的原型则放在 GRAPHICS.H 头文件中。常用的绘图函数如表 3—1 所示。

表 1-3 Turbo C 的绘图函数

函数名称	功 能
initgrah	启动绘图系统
closegraph	关闭绘图系统
grpahresult	取得绘图运算所发生的错误代码
registerbgidriver	通知绘图系统欲链结至程序的绘图驱动程序码
restorecrtmode	将屏幕重设为启动绘图系统前的状态
graphdefaults	重新设定全部的绘图参数
grapherrormsg	传回错误代码所对应的错误信息
setviewport	定义绘图窗口
setvisualpage	设定屏幕显示的显示作用页
setactivepage	设定绘图作用页

(续)

函数名称	功　　能
setgraphmode	切换不同的绘图模式
setgrahpbufsize	设定绘图函数所需的缓冲区大小
setallpalette	将整个调色板设置成指定的组合
setaspectratio	设定屏幕纵横比
setcolo	设定绘图颜色
setbkcolor	设定背景颜色
setfillpattern	设定用户自定的填图图样
setfillstyle	设定填图颜色及图样
setlinestyle	设定线条宽度和形式
setpalette	重新定义调色板中的某一个像素的值
setwritemode	设定画线时的写入模式
getcolor	返回目前绘图颜色
getbkcolor	返回背景颜色
getmaxcolor	返回目前的绘图模式下所允许的最大像素值
getgraphmod	取得目前的绘图模式
getdrivername	取得有关于绘图驱动程序名称的字串
getmaxmode	取得目前绘图驱动程序下所允许最大模式的编号
getmaxx	返回屏幕 x 坐标的最大值
getmaxy	返回屏幕 y 坐标的最大值
getmodename	取得指定模式编号的模式名称的的有关字串
getfillpattern	返回目前的填图的图样
getfillsettings	取得目前填色和填图图样的相关信息
getlinesettings	返回目前的线条形式和宽度
getarccoords	取得最近所画的弧的起点和终点
getaspectratio	返回屏幕纵横比
getdefaultpalette	返回目前驱动程序所设定的调色板结构
getmoderange	取得指定的绘图驱动程序允许的模式编号范围
getpalette	取得目前的调色板
getpalttesize	取得目前的调色板可用的颜色数
getviewsettings	取得与目前绘图窗口有关的信息
getx	返回目前窗口中的相对 x 坐标
gety	返回目前窗口中的相对 y 坐标
line	画线
lineto	由目前位置画一条线到指定点
lienrel	由目前位置到相对于目前位置一定距离的点之间画一直线
bar	画长条图
bar3d	画 3 度空间长条图
arc	画圆弧
circle	画圆

(续)

函数名称	功　　　能
ellips	画椭圆
rectangle	画矩形
drawpoly	将指定的点连成多边形
pieslice	画扇形
fillpoly	填满多边形
floodfill	填满封闭区域
fillellipse	画椭圆
getpixel	取得指定图点的颜色
putpixel	画点
getimage	将屏幕指定区块存入缓冲区中
putimage	将缓冲区内的图形显示于屏幕上
imagesize	计算使用 getimage 时所需要的内存大小
sector	画椭圆扇形
moveto	将绘图位置移到指定的一点
moverel	移动现行绘图位置 ·段指定距离
setviewport	设定屏幕一区块为绘图窗口
clearviewport	清除目前窗口
cleardevice	清除绘图屏幕，并将绘图位置移至屏幕左上角

为了让 Turbo C 在各种不同的显示适配卡上运用，Borland 公司提供了几种绘图驱动程序，以供程序员选用，它们的文件扩展名称都是 BGI(Borland Graphics Interface):

ATT.BGI

CGA.BGI

EGAVGA.BGI

HEEC.BGI

IBM8514.BGI

PC3270.BGI

如果使用单色绘图，必须有 HERC.BGI 这个文件，而在执行 EGA 或 VGA 绘图程序时，则要有 EGAVGA.BGI 文件，依此类推。以上驱动程序必须以 Turbo C 所提供的 BGIOBJ 公用程序将它们转换成.OBJ 文件之后，才能与绘图程序链结。

如果您使用的 Turbo C 是 1.5 版，那么在执行绘图程序之前，必须先产生一个工程文件(Project file)，扩展名是 PRJ，其内容为:(GRAPROG.PRJ)

graprog.c

graphcis.lib

然后在集成环境下，选取 Project 中的 Project Name 选项，指定为刚才产生的工程文件 GRAPROG.PRJ 即可。

如果您使用的是 Turbo C 2.0 版，则只须在集成环境下，选取 Option 中的 Linker

选项，将 graphics 项设定为 ON 即可。

若是使用命令行编译器 TCC.EXE，不论是 1.5 或 2.0，都必须在程序行后再加上图形程序库，如下所示：

```
tcc gaprog.c graphics.lib
```

环境设定完成后，在程序中便可以使用 initgraph 函数说明进入图形模式：

函数：initgraph()

原型：void far initgrapn (int far *driver ,int far *made, char far *path);

此一函数需要 3 个 int far 形态的参数：

1. driver 用来指定采用哪一个绘图驱动程序，它有下列 11 种选择：

表 1-4　绘图驱动程序的选择常数

常数	数值
DETECT	0
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

这些常数在GRAPHICS.H头文件中定义成0到10的对应数值，例如要执行VGA绘图程序时，driver 参数应该设定为 VGA 或者 9。如果将 driver 的值设定为 DETECT，则系统将会自动侦测目前所连接的显示适配器，而选用适当的绘图驱动程序。

2. mode：图形模式。表1–5所列的是Turbo C的图形模式。在表中可以看出：一个驱动程序都有数种图形模式可供选用，我们可利用 mode 参数来选择不同的模式。如果要执行 640×480 16 色的绘图程序，则 mode 参数应该设定为 EGAHI 或者 2。

3. path：代表绘图驱动程序所在的文件路径。假如path的值为NULL或者“”时，表示绘图驱动程序在当前的工作日录之中。

表 1-5　Trubo C 的绘图模式

驱动程序	模式	数值	分辨率／色彩／页数
CGA	CGAC0	0	320×200 调色板 0，1 页
	CGAC1	1	320×200 调色板 1，1 页
	CGAC2	2	320×200 调色板 2，1 页
	CGAC3	3	320×200 调色板 3，1 页
	CGAHI	4	640×200　2 色，1 页
MCGA	MCAGAC0	0	320×200 调色板 0，1 页
	MCGAC1	1	320×200 调色板 1，1 页
	MCAGC2	2	320×200 调色板 2，1 页
	MCAGC3	3	320×200 调色板 3，1 页
	MCGAMED	4	640×200　2 色，1 页
	MCGAHI	5	640×480　2 色，1 页
EGA	EGALO	0	640×200　16 色，4 页
	EGAHI	1	640×350　16 色，2 页
EGA64	EGA64LO	0	640×200　16 色，1 页
	EGA64HI	1	640×350　4 色，1 页
EGAMONO	EGAMONOHI	3	640×360　4 页
HERC	HERCMONOHI	0	720×348　2 页
ATT400	ATT400C0	0	320×200 调色板 0，1 页
	ATT400C1	1	320×200 调色板 1，1 页
	ATT400C2	2	320×200 调色板 2，1 页
	ATT400C3	3	320×200 调色板 3，1 页
	ATT400CMED	4	640×200　1 页
	ATT400CHI	5	640×400　1 页
VAG	VGALO	0	640×200　16 色，4 页
	VGAMED	1	640×350　16 色，2 页
	VGAHI	2	640×480　16 色，1 页
PC3270	PC3270HI	0	720×350　1 页

在执行 initgraph 函数时，如果参数 driver 的值指定为 DETECT，则 initgraph 中第一个参数所返回的值就是目前显示卡所使用的驱动程序的代号，而第二个参数便会自动调整为该驱动程序所能提供的最高分辨率的模式。使用 detectgraph 函数也可达到此项功能。利用这项功能，就可以写一个程序来检测目前计算机所具有的显示适配器是属于何种形式。两个程序分别用 Initgraph 及 detectgraph，介绍如下：

程序 1

```
#include"graphics.h"
```

```
#include "stdio.h"
main()
{
        int graph__driver,graph__mode;

        detectgraph(&graph__driver,&graph__mode);
        switch(graph__driver)  {
                case VGA       : puts("There is a VGA card.");
                                 break;
                case EGA       : puts("There isa EGA card.");
                                 break;
                case HERCMONO: puts("There is a Hercules card.");
                                 break;
                }
}
```

程序 2

```
#include "graphics.h"
#include "stdio.h"
main()
{
     int graph__driver=DETECT;
     int graph__mode;

     intigraph(&graph__driver;&graph__mode,"");
     switcn(graph__driver) {
            case VGA        : outtextxy(250,240,"It's in VGA mode.");
                              break;
            case EGA        : outtextxy(250,240,"It's in EGA mode.");
                              break;
            case HERCMONO : outtextxy(250,240,"It's in Hercules mode.");
                              break;
     }
}
```

使用 initgraph 和 detectgraph 的不同点在于 initgraph 是判断后即进入绘图模式，而 detectgraph 则只有判断绘图适配器，并未进绘图模式。所以若是程序只需侦测显示卡，则以使用 detectgraph 较为恰当。

本节最后再将绘图程序所必须的部分罗列于后，让读者有个更清晰的概念。

```
#include<graphics.h>
   .
   .
   .
```

```
main()
{
        int  graph__driver = DETECT,graph__mode;
        init graph(&graph__driver,&graph__mode,"");
                .
                .
                .
                closegraph();
}
```

第二章　图形显示技巧

2.1　概　述

在简报系统(Presentation System)、CAI、计算机游戏等软件中，经常会看到各种图形显示技巧。凭着图形的移动、交错、雨状、百叶窗、积木等显现方式，使画面变得更为生动活泼、更能吸引观众。在本章中，我们所要探讨的就是目前各种绘图软件中最常见的几种图形显示技巧。

我们首先介绍图形显示的基本原理，接着探讨各种绘图模式在图形显示上的可行性，再进述各种适配器（Hercules，EGA，VGA）所需注意的事项。有了以上的基础知识后，就可以解释几种具有代表性的技巧算法了，最后再以一个完整的实例做为本章的结束。这个实例 Turbo C 编写完成，希望它能提供你许多的灵感，做为以后开发相关的应用软件之用。

2.2　图形显示基础知识

2.2.1　图形显示基本原理

最简单的图形显示方法，就是直接将图形显示在屏幕上，不过这种方法却毫无变化，且毫无技巧可言。如果要让图形展示具有震撼性效果，我们将采取下列步骤进行:

1. 先将图形由磁盘上载入到内存的一块缓冲区中，以避免在显示的过程中，因磁盘的速度太慢而使效果大打折扣。
2. 再将缓冲区中的图形数据，使用特殊的技巧搬移到屏幕所相对应的显示内存上。

如此，图形就会以特殊的效果展现在屏幕上了。

在此，将上述的原理用图形表示，请参考图 2-1。在这里有许多必须具备的基础知识及注意事项，分成三部分说明于后。

1. 将磁盘上的图形数据读入缓冲区中。
2. 使用搬移指令，将缓冲区的数据搬至适配器缓冲区中。
3. 在数据搬至显示内存的同时，屏幕也会立即显示。

2.2.1.1　图形数据的装入:

图形显示所遇到的第一个问题便是如何将图形数据装人存储器缓冲区中。当你得到一个图形文件时，你所面对的第一件事就是：并非每一种图形文件的储存格式都相同，随着该图形的文件扩展名的不同，可能其储存的方法也跟着不同，而且原图形数据可能经过压缩处理，所以储存的方法和读取的方法必须配合，才能顺利地显示原来的图形数据。关于这一点，我们在第三章有详细的说明。

图 2-1 图形显示的步骤

2.2.1.2 绘图模式的选择:

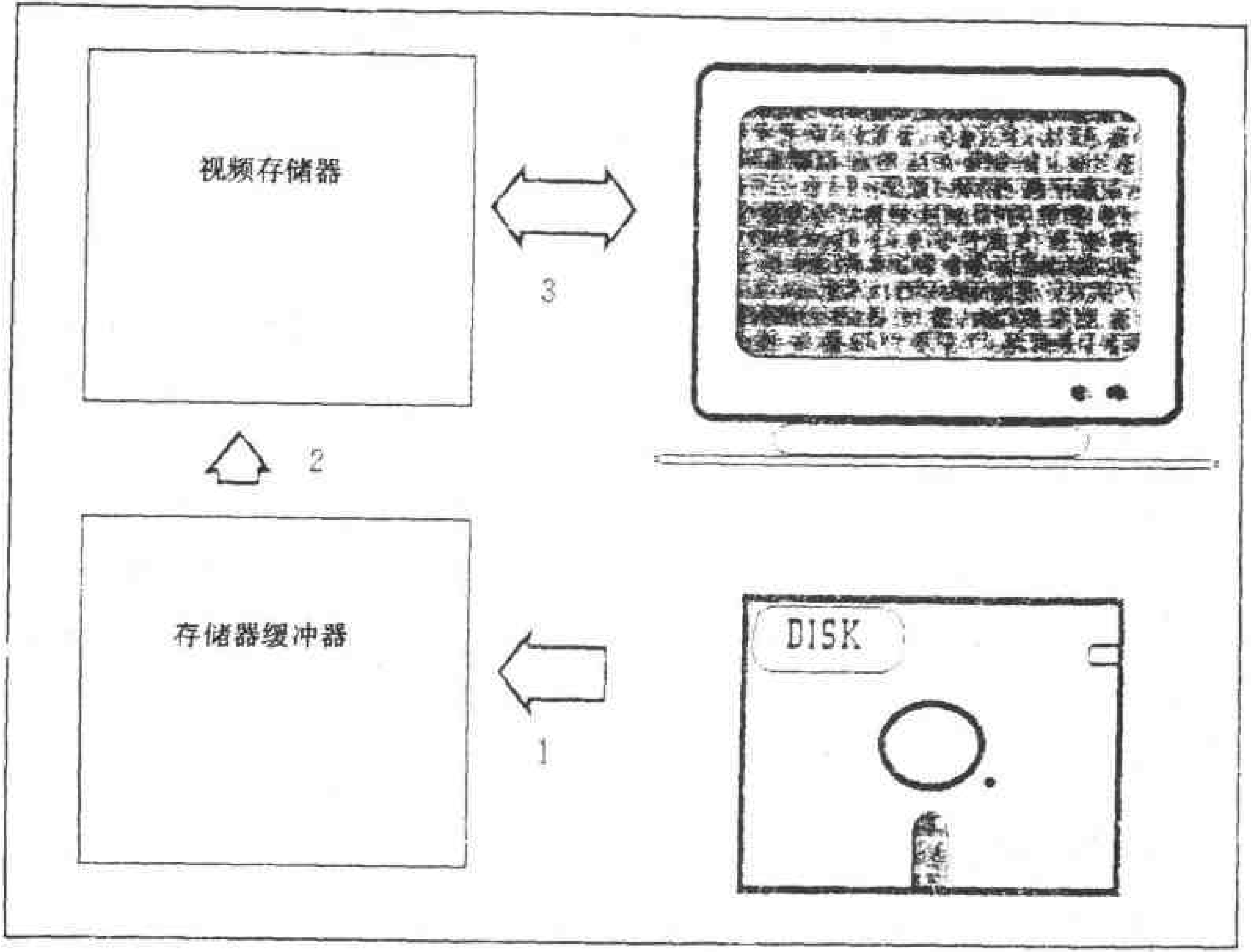
了解图形的储存的信息后，接下来所关心的便是读入的图形数据放在哪里．当然不是直接就放到屏幕上去（正确地说，应该是屏幕所对应的显示内存)．如前所说（图 2-1)，必须先行放在缓冲区中．但是需要多大的缓冲区呢？这得视图形的分辨率和颜色来决定．以全屏幕的图形来计算：720×348 单色图形需要 31320 字节，640×350 16 色图形需要 112000 字节，而 320×200 256 色图形则需要 64000 字节．如此大的内存如果直接由主存中配置并不合算，所以必须要另外找一块合适的空间．很幸运的是，许多适配器在几个特定的绘图模式下都提供了能够储存一页以上的屏幕图形数据所需的显示内存，只要使用这几个绘图模式，就可以将图形数据载入另一页尚未用到的显示内存中，如此，内存的问题便可轻易地解决了．

表 2-1 便是 IBM PC BIOS 所提供的绘图模式的相关数据，凡是显示内存的页数在 2 页以上者，都相当适合于图形显示使用，而假若只有 1 页显示内存，就需要较复杂的手续才能达到目标．

2.2.1.3 图形数据的搬移:

图形数据载入缓冲区后，再下来就是搬移数据了．在 C 语言中提供了一个强有力的数据搬移函数 movedata，以下是对这个函数的介绍:

函数：movedata()


视频存储器
3
2
存储器缓冲器
DISK
1

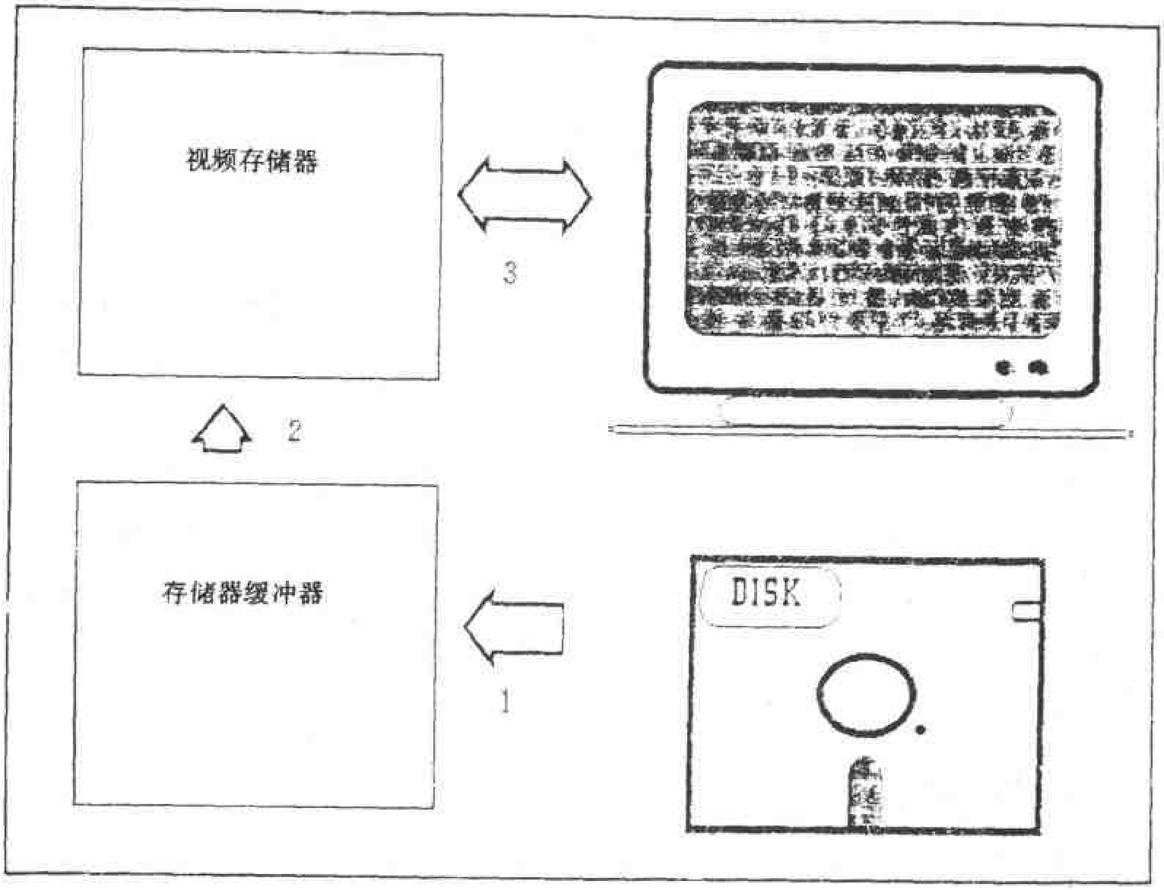


图 2–1　图形显示的步骤

2.2.1.2　绘图模式的选择:

了解图形的储存的信息后，接下来所关心的便是读入的图形数据放在哪里．当然不是直接就放到屏幕上去（正确地说，应该是屏幕所对应的显示内存)。如前所说（图 2–1)，必须先行放在缓冲区中。但是需要多大的缓冲区呢？这得视图形的分辨率和颜色来决定。以全屏幕的图形来计算：720×348 单色图形需要 31320 字节，640×350 16 色图形需要 112000 字节，而 320×200 256 色图形则需要 64000 字节。如此大的内存如果直接由主存中配置并不合算，所以必须要另外找一块合适的空间．很幸运的是，许多适配器在几个特定的绘图模式下都提供了能够储存一页以上的屏幕图形数据所需的显示内存，只要使用这几个绘图模式，就可以将图形数据载入另一页尚未用到的显示内存中，如此，内存的问题便可轻易地解决了。

表 2–1 便是 IBM PC BIOS 所提供的绘图模式的相关数据，凡是显示内存的页数在 2 页以上者，都相当适合于图形显示使用，而假若只有 1 页显示内存，就需要较复杂的手续才能达到目标。

2.2.1.3　图形数据的搬移:

图形数据载入缓冲区后，再下来就是搬移数据了。在 C 语言中提供了一个强有力的数据搬移函数 movedata，以下是对这个函数的介绍:

函数：movedata()

原型: movedata(int segrc ,intoffsrc,int segdest,
int offdsct,undigned numbytes);

功能: 在地址之间搬移 N 字节的数据

返回值: 无

说明: 在显示内存之间的数据搬移是属于远程数据搬移，而 movedata 正符合了这个要求.

表 2-1 PC BIOS 所提供显示模式中有关绘图的部分

模式	分辨率	色彩	显示内存的起始地址
4	320×200	4	B800
4	320×200	4	B800
6	640×200	2	B800
D	320×200	16	A000
E	640×2000	16	A000
F	640×350	3	A000
10h	640×350	16	A000
11h	640×480	2	A000
12h	640×480	16	A000
13h	320×200	256	A000

表 2-2 Turbo C 2.0 所提供的绘图模式（部分）

图形驱动器	图形模式	分辨率	色彩	页数
EGA	EGALO	640×200	16	4
	EGALI	640×350	16	2
HERCMONO	HERCMONOHI	720×348	2	2
VGA	VGALO	640×200	16	2
	VGAMED	640×350	16	2
	VGAHI	640×480	16	1

2.2.2 各种绘图适配器概述

在第一章中，曾经概略地介绍了各种显示适配器的开发状况，本节则就显示适配器与程序设计直接相关的部分，做更进一步的探讨。

2.2.2.1 Hercules 绘图适配器

Hercules 绘图适配器是目前 PC 上最受欢迎的高分辨率绘图适配器，其最高分辨率为 720×348，提供 2 页显示内存，内存的地址起点分别为 B000:0000 及 B800:0000。基于硬件设计本身的考虑，这种适配器的图形数据储存并不呈线性排列，而是以四条扫描线为一组，分别从 B000:0000，B000:2000，B000:4000 及 B000:6000 开始储存:

第一条扫描线的数据储存地址　B000:0000～B000:0059

第二条扫描线的数据储存地址　B000:2000～B000:2059

第三条扫描线的数据储存地址　B000:4000～B000:4059

第四条扫描线的数据储存地址　B000:6000～B000:6059

而第五条扫描线从 B000:005A 开始到 B000:00B3...依此类推。其屏幕扫描线与地址对应映射如图 2-2 所示:

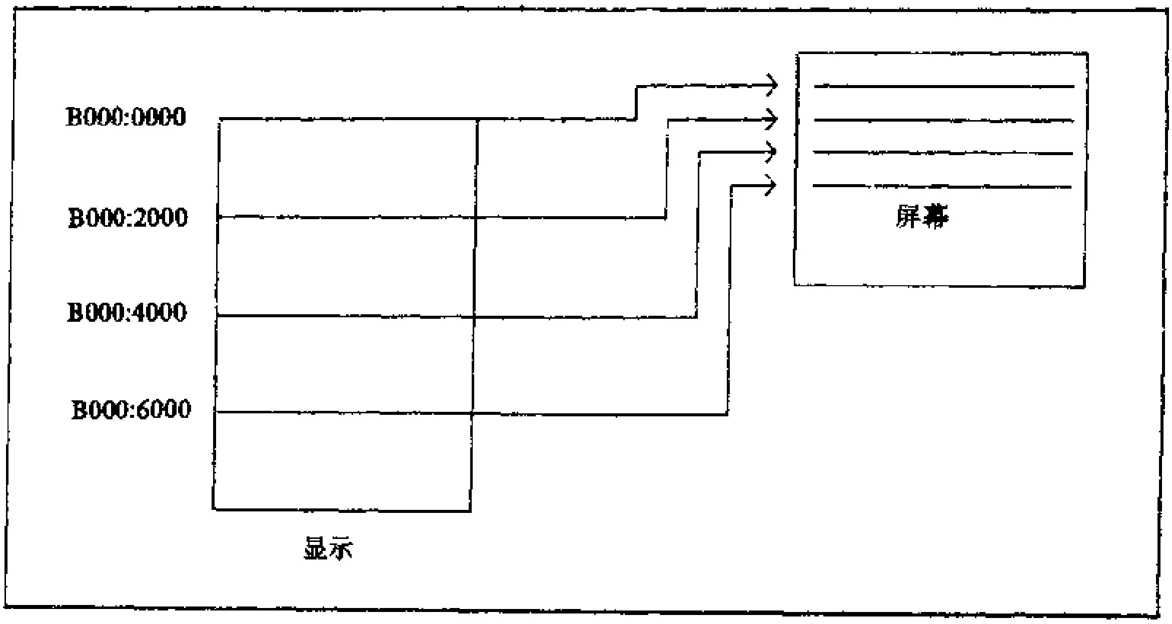


图 2-2　Hercules 绘图内存适配器与屏幕的对应图

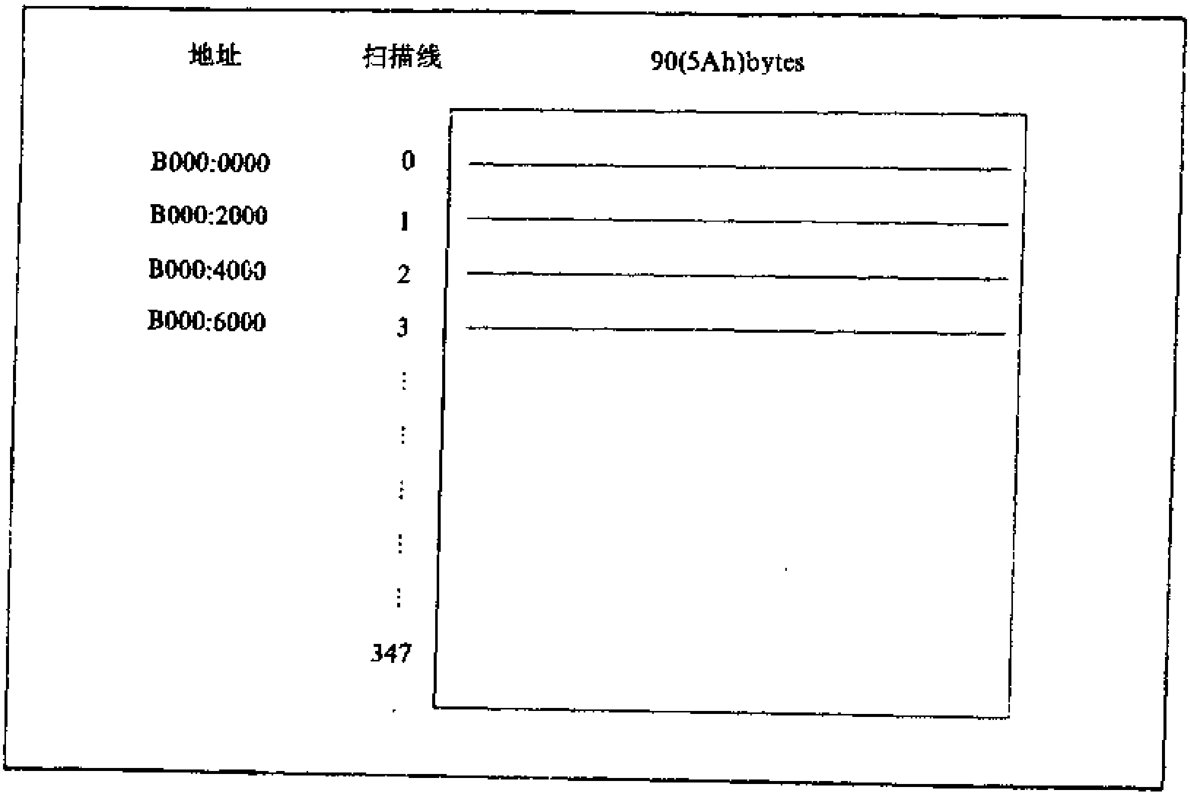


图 2-3　单色屏幕的逻辑结构

此种适配器在屏幕上每一点(Pixel)以一个 bit 来表示，所以 720 点共需要 720 bit 即 720／8＝90bytes，也就是每一条扫描线共占 90 个字节的图形数据。

在图形显示过程中，常常有将第 1 页的第 m 条扫描线的图形数据搬移至第 0 页的 n 条扫描线的情形，遇此情形，必须先将扫描线的编号换算成内存地址才行。在换算前，请先参考图 2–3 的屏幕逻辑结构，可以得到更确切的了解，在往后的内存搬移中也是以 2–3 图为结构进行讨论。

依据图 2–3 的逻辑结构，扫描线第 0 条的起始地址为 B000:0000（第 0 页），而第 N 条的起始地址应为:

第 0 页:

B000:(N%4) * 0x2000+(N／4) * 0x5A

第 1 页:

B800:(N%4) * 0x2000+(N／4) * 0x5A

如果要将第 1 页的第 M 条扫描线搬移至第 0 页的第 n 条扫描线，那么就可以写成:

```
movedata(0xB800,(M%4) * 0x2000+(M/4) * 0x5A,
        0xB000,(N%4) * 0x2000+(N/4) * 0x5A,90);
```

然而在图形显示中，一个很重要的考虑因素就是速度，所以在实际应用上，必须使用查表法来达到这项要求，也就是使用一次大小为 348 的数组储存每条扫描线的起始地址，届时只要直接取出即可，省去了一边搬移一边换算的麻烦。制作换算数组的函数如程序 2–1 所列，有了这个函数后，上面的例子便可写成:

```
movedata(oxb800,addr[m],0xb000,addr[n],90);
```

程序显得既快而又简洁。

程序 2–1

```
unsigned int addr[350];
map2()
{
      int i;
      for(i=0,i<348;i++)
            addr[i]=0x2000 * (i% 4)+0x5a( * i/4);
}
```

2.2.2.2 EGA 绘图适配器:

EGA 绘图适配器所提供的最高分辨率为 640×350 16 色的绘图模式。如果 EGA 卡的显示内存在 256K 以上，则此种模式就有 2 页显示页可做为图形显示之用。此种适配器的起始地址分别是:

第 0 页　A000:0000

第 1 页　A800:0000

它在硬件结构上是采用位平面(bit plane)的排列方式，这种排列方法较为复杂，并不属本书的讨论范围内。在图形显示时，只要知道每当于搬移显示内存之前必须先执行:

```
outportb(0x3ce,5);
```

outportb(0x3cf,1);

才能顺利搬移．同时，搬移指令也不能沿用 movedata，而必须另外写一个函数才能顺利执行．这个函数如程序 2–2 所示，它是以汇编语言写成的，但用法却和 movedata 一模一样。

程序 2–2

```
            .MODEL      SMALL
            .CODE
            public      __cmove
__cmove     proc
            push        bp
            mov         bp,sp
            push        es
            push        ds
            push        di
            push        si
            push        ax
            push        cx
            push        dx
            mov         ax,[bp+4]
            mov         ds,ax
            mov         ax,[bp+6]
            mov         si,ax
            mov         ax,[bp+8]
            mov         es,ax
            mov         ax,[bp+10]
            mov         di,ax
            mov         cx,[bp+12]
rep         movsb
            pop         dx
            pop         cx
            pop         ax
            pop         si
            pop         di
            pop         ds
            pop         es
            pop         bp
            ret
__cmove     endp
            END
```

编译方法如下：

```
C:\MASM>masm/mx comve
Microsoft (R) Macro Assembler Version 5.00
```

```
Copyright (C) Microsoft Corp 1981-1985,1987.  All rights reserved.
  51528 + 397592 Bytes symbol space free
      0 Warning Errors
      0 Severe Errors

C:\MASM>
```

汇编时请记住加上/mx参数。关于本函数的使用方法，将于往后应用时再予详述。

图 2-4 是 640×350 16 色的屏幕逻辑结构。基于速度上的考虑，这个模式的扫描线的内存地址虽然成线性排列，仍然需要一个查表换算用的数组，而程序 2-3 便是制作查表数组的函数 map16。

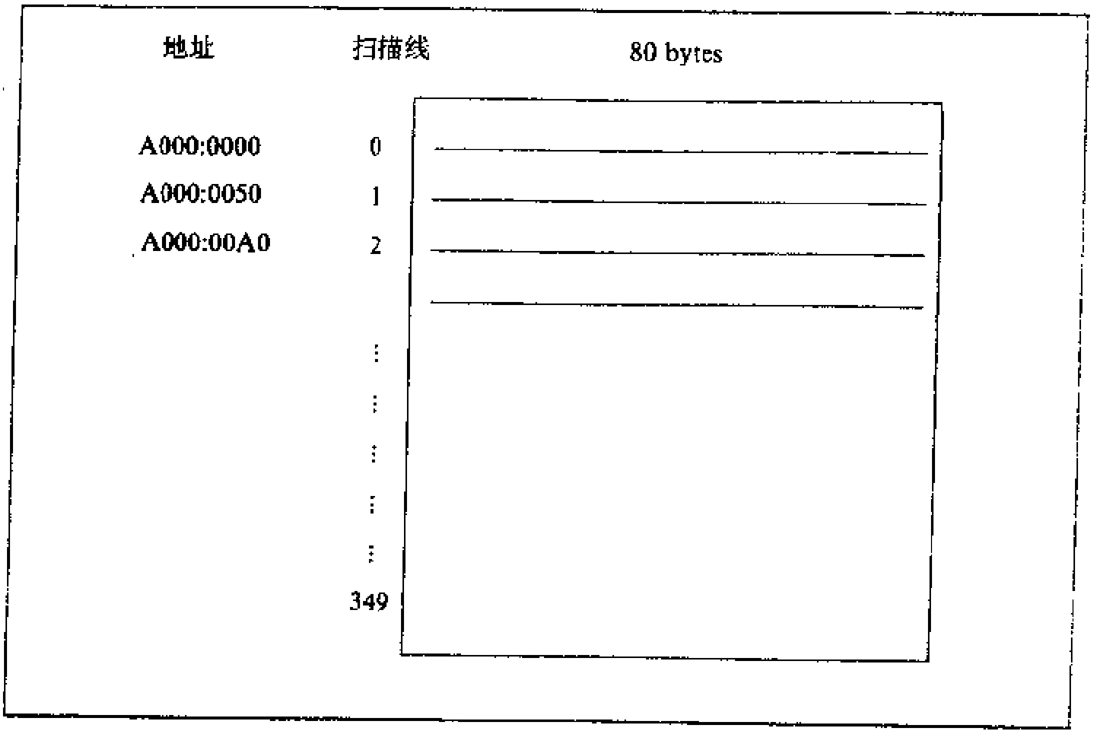


图 2-4　640×350　16 色屏幕逻辑结构

程序 2-3 map16.C

```
        unsigned int addr[350];
        map16()
{
                int i;
                for(i=0;i<350;i++0
                    addr[i]=i * 80;
}
```

2.2.2.3 *VGA* 绘图适配器

VGA 绘图适配器提供了许多绘图模式，包括上述的 640×350 16 色模式。其中比较特殊的是 320×200 256 色的绘图模式。在此模式之下，屏幕上可同时出现多达 256 种颜

色，造成相当逼真的视觉效果，只可惜分辨率太低，并且只提供 1 页的显示内存，如果要做特殊效果显示，则需要一些较复杂的手续才能完成。在此仅讨论其基本结构，至于效果显示则留等后面的章节再介绍。

这个绘图模式的屏幕逻辑结构如图 2–5 所示。它的内存起始地址为 0A000:0000，扫描线和内存地址呈线性排列。每一个 byte 表示一个像素(pixel)，因为 1byte = 8bits，所以共可以表示 $2^8 = 256$ 种颜色。

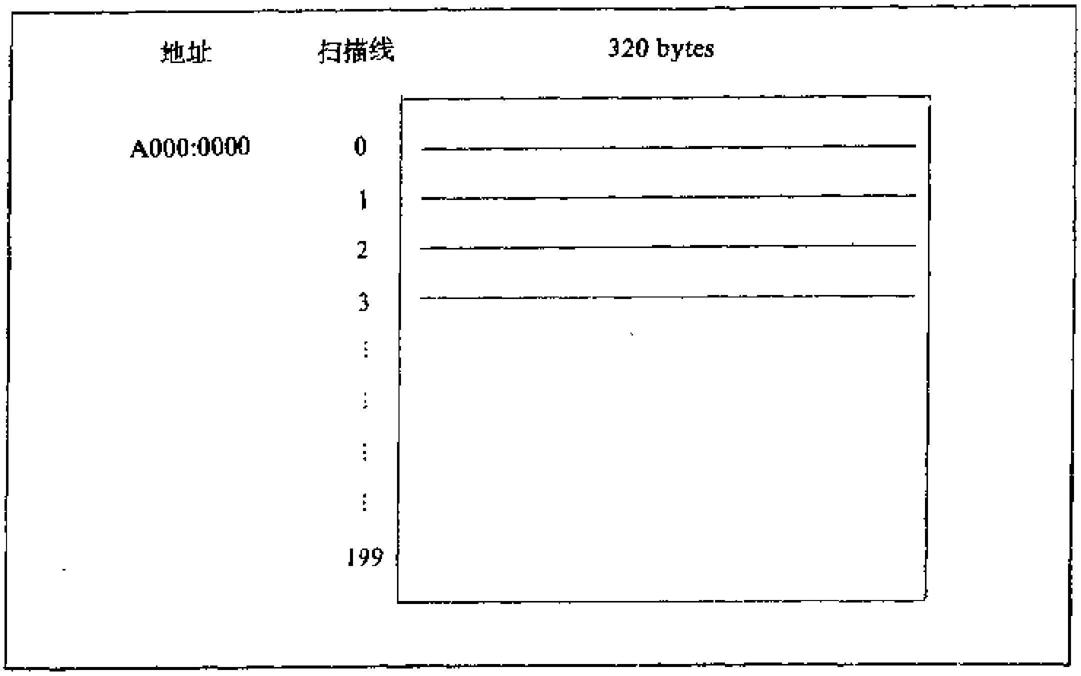


图 2–5　320×200　256 屏幕逻辑结构

此外 VGA 适配器还提供另一种 640×480　16 色的绘图模式，由于它只提供了 1 页的显示内存，因此而不属本章的范围之内。

2.3　各种图形显示方法

2.3.1　推拉效果

推拉效果就是以屏幕为中心，将要显示的图形由上、下、左、右任一方向从屏幕外拉进屏幕内显示，同时将原来的屏幕上的旧图盖掉。此种效果可分为四种：即上拉、下拉、左拉、右拉等等，但是原理都差不多，在此仅以上拉效果为例加以说明。

图 2–6 所示即为上拉效果示意图。本效果首先将放在暂存页图形的第一条扫描线(扫描线 0)，搬移至可视页之最后一条扫描线（扫描线 347)，接着再将暂存页的前二条扫描线（扫描线 0，1)，依序搬移可视页的最后二条扫描线（扫描线 346，347)，然后再搬移第三条扫描线，第四条扫描线……直到全部所有图形数据为止。在搬移的过程中即可看到要显示的图形由屏幕的下方逐渐地浮上，而达到上拉的效果。

而下拉效果原理也是如此，只不过方向相反罢了。程序 2–4 就是为上拉效果的算法。

程序 2-4

```
FOR I=347 TO 0 STEP -1
  FOR J=0 TO 347 -1
   MOVELINE Page1(J) TO Page0(I+J)
  NEXT J
NEXT I
```

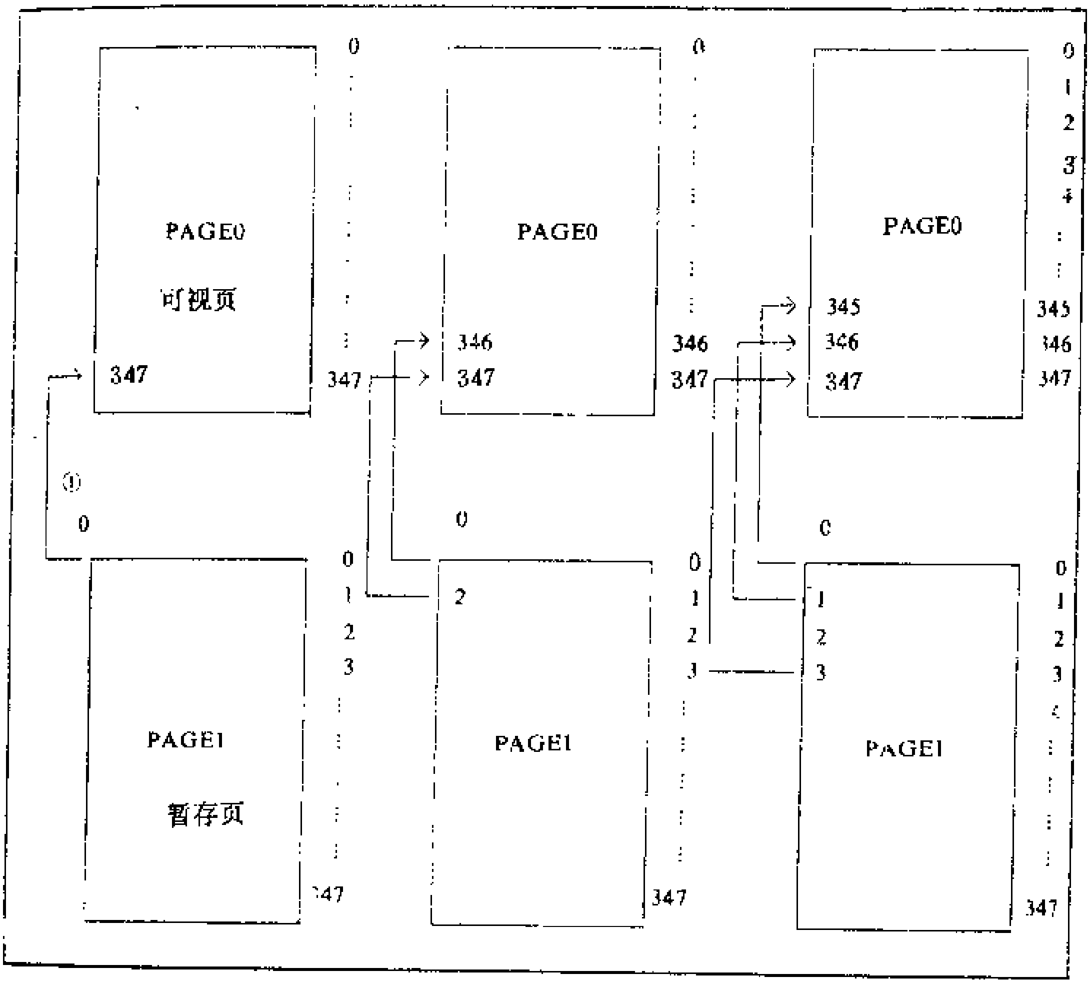


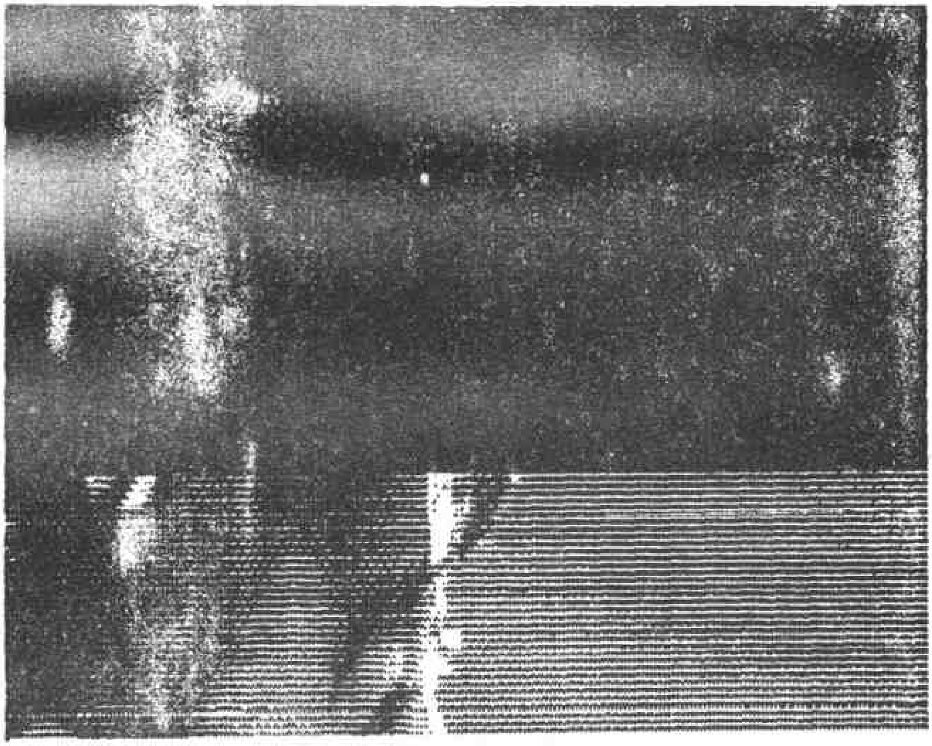
图 2-6 推拉效果示意图

以下就是上拉效果程序 PULL.C 的完整列表:

```
/*  PULL.C  */

#include <graphics.h>
#include <stdio.h>
#include <alloc.h>

void loadmapm(char *);
void map2(void);
```

```
void pull_up2(void);

unsigned addr[348];

main()
{
      int graph_driver=HERCMONO, graph_mode=HERCMONOHI;

      initgraph(&graph_driver, &graph_mode, "");      /* Set graphics mode */
      loadmapm("demo.map");           /* Load a picture into video page 1  */
      map2();                         /* Tabulate
      pull up2();                     /* Show the picture using "pull_up" effect */
      getch();                        /* Wait for pressing key state */
      closegraph();                   /* Close graphics mode */
}

void loadmapm(char *name)
{
              FILE  *fp;
              char  *buf;
              char low, high;
              unsigned size, x, y;
              union {
          char ch[2];
               int no;
              } value:

             if ((fp=fopen(name,"rb")) == NULL)       {
                    puts("\a");
                    closegraph();
                    printf("\nFile reading error!!");
                    exit(1);
             }
             else      {
                 if ((unsigned)(filelength(fileno(fp))) >= coreleft())
                 {
                      puts("\a");
                      closegraph();
                      printf ("\nSorry! Memory is not enough.\n");
                      exit(1);
                 }
                 else
                      size=filelength(fileno(fp));
                 buf=(char *) malloc(size);
```

```
                setvisualpage(0);
                setactivepage(1);
                fread(buf,1,size,fp);
                value.ch[0] = buf[7];
                value.ch[1] = buf[8];
                x = value.no;
                value.ch[0] = buf[9];
                value.ch[1] = buf[10];
                y = value.no;
                x = (720-x)/2;
                y = (348-y)/2;

                clearviewport();
                buf[7]--; buf[9]--;
                putimage(x,y,buf+7,COPY_PUT);
                fclose(fp);
                free(buf);
            }
    }

void map2()
{
     register int i, j;

     for(i=0; i<348; i++)
          addr[i] = 8192 * (i%4) + 90 * (i/4);
}

void pull_up2()
{
     int i, j;

     for(i=347; i>=0; i--)
          for(j=0; j<348-i; j++)
               movedata(0xb800, addr[j], 0xb000, addr[i+j], 90);
}
```



(1)

(2)

图 2-7　推拉效果程序 PULL.C 的执行结果

2.3.2　垂直交错效果

垂直交错效果为：将一张要显示的图形拆成两部分，奇数条扫描线的部分由屏幕的上端往下呈现，而偶数条扫描线的部分则由屏幕的下端往上逐渐呈现。若两动作同时进行，则从屏幕上便可看到分别由上下两端出现的较淡图形向屏幕中央移动，直到完全清楚为止。

本效果的主要着眼点在于奇偶数扫描线必须分别搬移：奇数扫描线由上往下搬移而偶数扫描线由下往下搬移，而且二者必须同时进行。仔细分析，其实际效果是结合上拉及下拉效果而来的，只是在上拉及下拉时，仅能分别拉奇、偶条扫描线罢了。

接着我们来看看垂直交错效果的算法及其相对应的程序：

```
算法
FOR I=0 TO 346 STEP 2
  FOR J=I TO 0 STEP -2
   MOVELINE Page1(346 -I+J) TO Page0(J)
   MOVELINE Page1(I-J+1) TO Page0(347-J)
  NEXT J
NEXT I
```

(3)

(3)

图 2–7　推拉效果程序 PULL.C 的执行结果

2.3.2　垂直交错效果

垂直交错效果为：将一张要显示的图形拆成两部分，奇数条扫描线的部分由屏幕的上端往下呈现，而偶数条扫描线的部分则由屏幕的下端往上逐渐呈现。若两动作同时进行，则从屏幕上便可看到分别由上下两端出现的较淡图形向屏幕中央移动，直到完全清楚为止。

本效果的主要着眼点在于奇偶数扫描线必须分别搬移：奇数扫描线由上往下搬移而偶数扫描线由下往下搬移，而且二者必须同时进行。仔细分析，其实际效果是结合上拉及下拉效果而来的，只是在上拉及下拉时，仅能分别拉奇、偶条扫描线罢了。

接着我们来看看垂直交错效果的算法及其相对应的程序：

```
算法
FOR I=0 TO 346 STEP 2
  FOR J=I TO 0 STEP -2
   MOVELINE Page1(346 -I+J) TO Page0(J)
   MOVELINE Page1(I-J+1) TO Page0(347-J)
  NEXT J
NEXT I
```

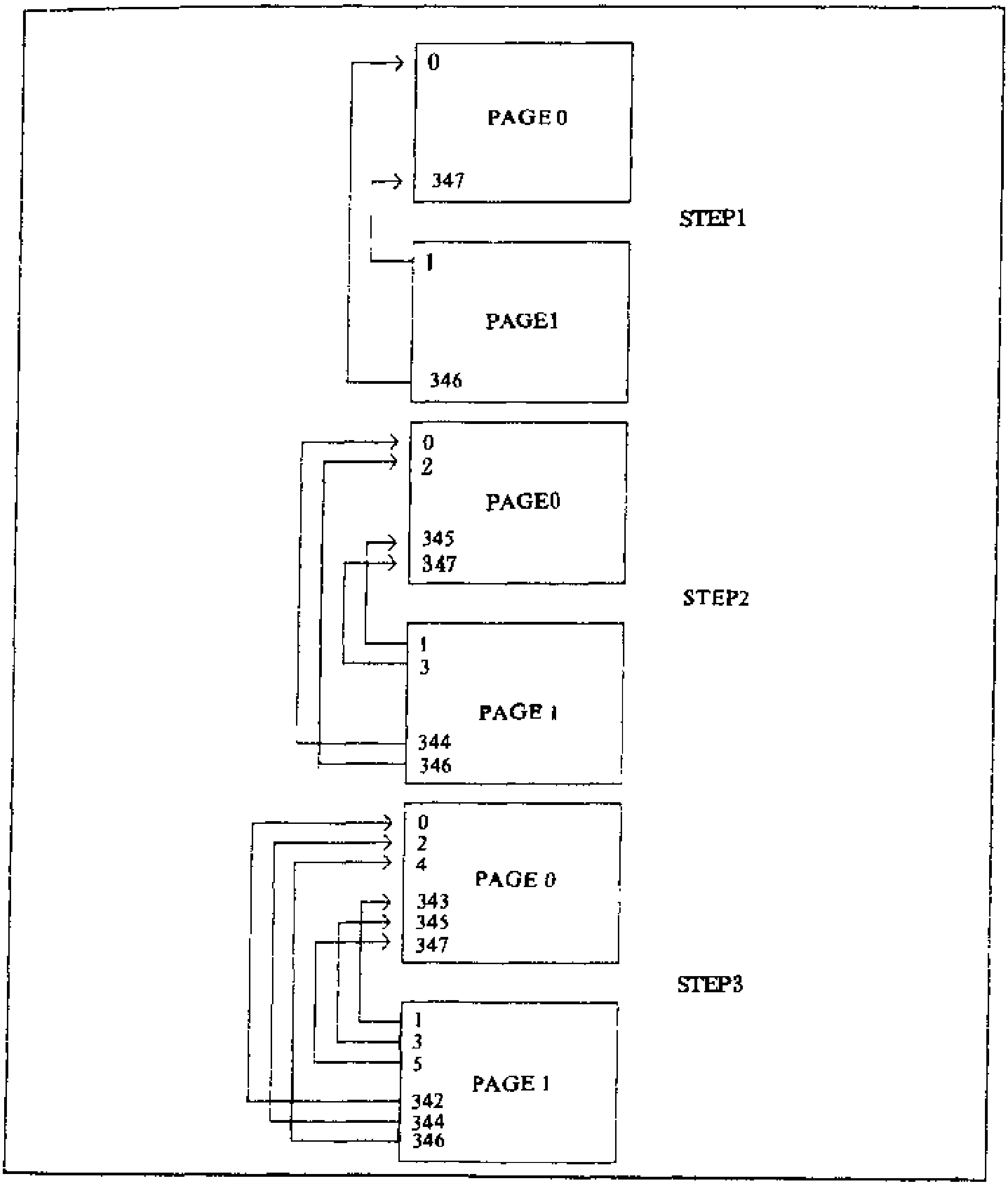


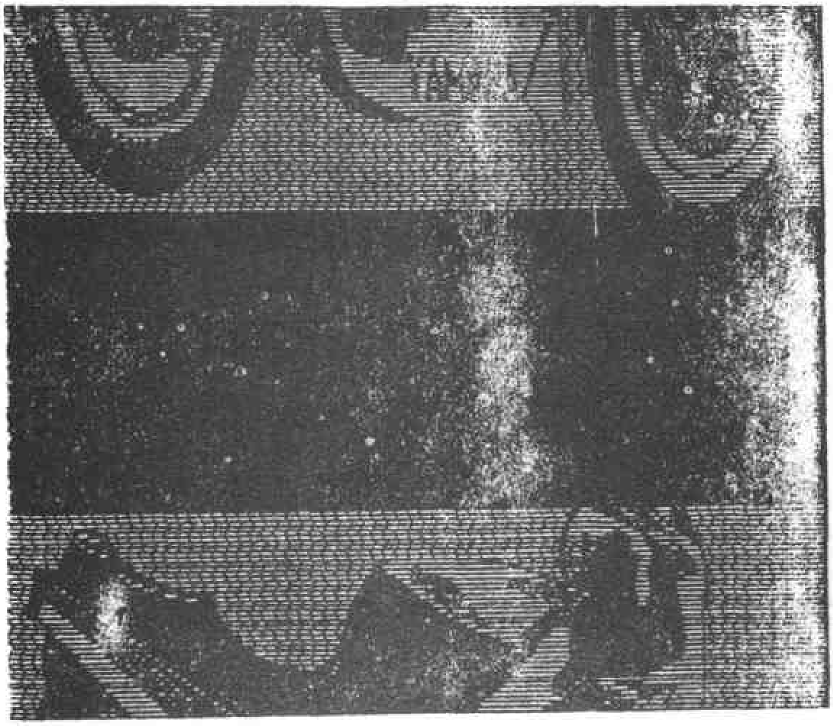
图 2-8 垂直交错效果示意图

垂直交错效果程序 CROSSV.C

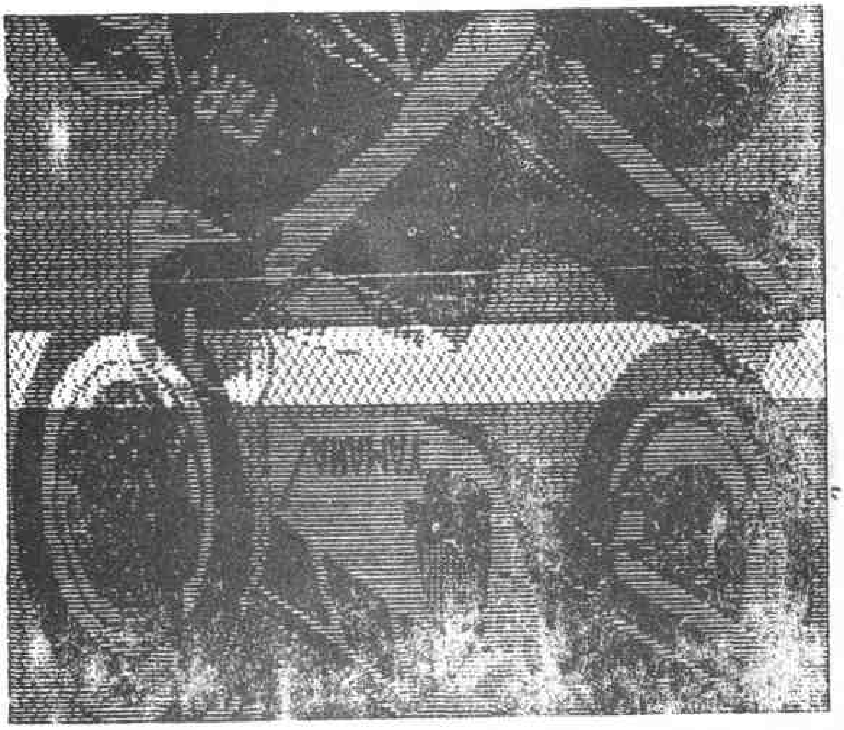
```
/* CROSSV.C */

#include <graphics.h>
#include <stdio.h>
#include <alloc.h>
```

```
void loadmapm(char *);
void map2(void);
void cross_v2(void);

unsigned addr[348];

main()
{
      int graph_driver=HERCMONO, graph_mode=HERCMONOHI;

      initgraph(&graph_driver, &graph_mode, ""); /* Set graphics mode */
      loadmapm("demo.map");     /* Load a picture into video page 1  */
      map2();                   /* Tabulate */
      cross_v2();               /* Show the picture using "cross_v" effect */
      getch();                  /* Wait for pressing key state */
      closegraph();             /* Close graphics mode */
}

void loadmapm(char *name)
{
      FILE  *fp;
      char  *buf;
      char low, high;
      unsigned size, x, y;
      union {
            char ch[2];
            int no;
      } value;

      if ((fp=fopen(name,"rb")) == NULL)    {
            puts("\a");
            closegraph();
            printf("\nFile reading error!!");
            exit(1);
      }
      else     {
        if ((unsigned)(filelength(fileno(fp))) >= coreleft()) {
              puts("\a");
              closegraph();
              printf ("\nSorry! Memory is not enough.\n");
              exit(1);
        }
        else
```

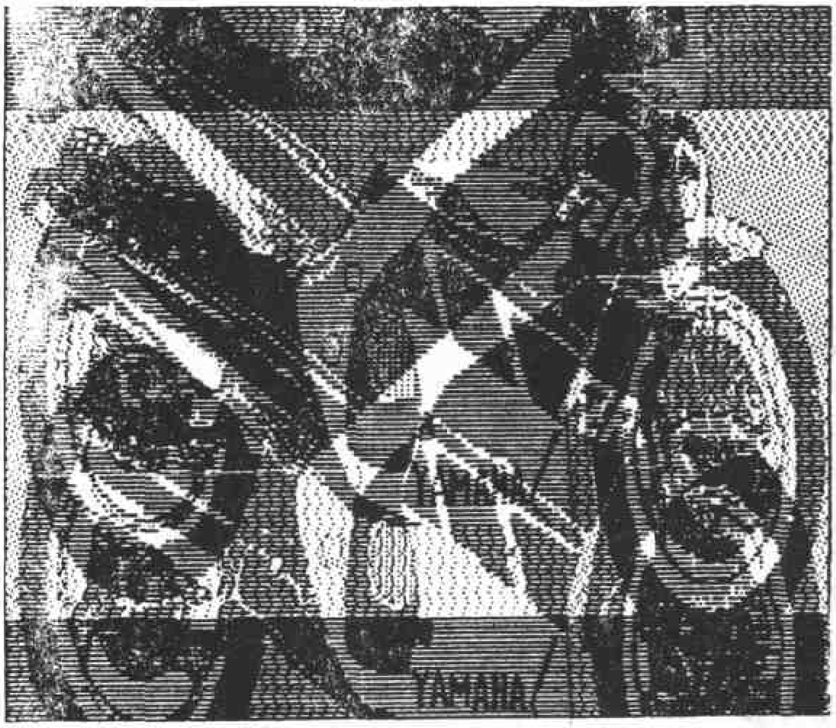
```
            size = filelength(fileno(fp));
        buf = (char  * ) malloc(size);
        setvisualpage(0);
        setactivepage(1);
        fread(buf,1,size,fp);
        value.ch[0]  =  buf[7];
        value.ch[1]  =  buf[8];
        x  =  value.no;
        value.ch[0]  =  buf[9];
        value.ch[1]  =  buf[10];
        y  =  value.no;
        x  =  (720-x) / 2;
        y  =  (348-y) / 2;

        clearviewport();
        buf[7]--; buf[9]--;
        putimage(x,y,buf+7,COPY_PUT);
        fclose(fp);
        free(buf);
    }
}

void map2()
{
    register int i, j;

    for(i=0; i<348; i++)
        addr[i]  =  8192 * (i%4)  +  90 * (i / 4);
}

void cross v2()
{
    int i,j;

    for(i=0;i<347;i+=2)
        for(j=i;j>=0;j-=2)        {
            movedata(0xb800,addr[346-i+j],0xb000,addr[j],90);
            movedata(0xb800,addr[i-j+1],0xb000,addr[347-j],90);
    }
}
```



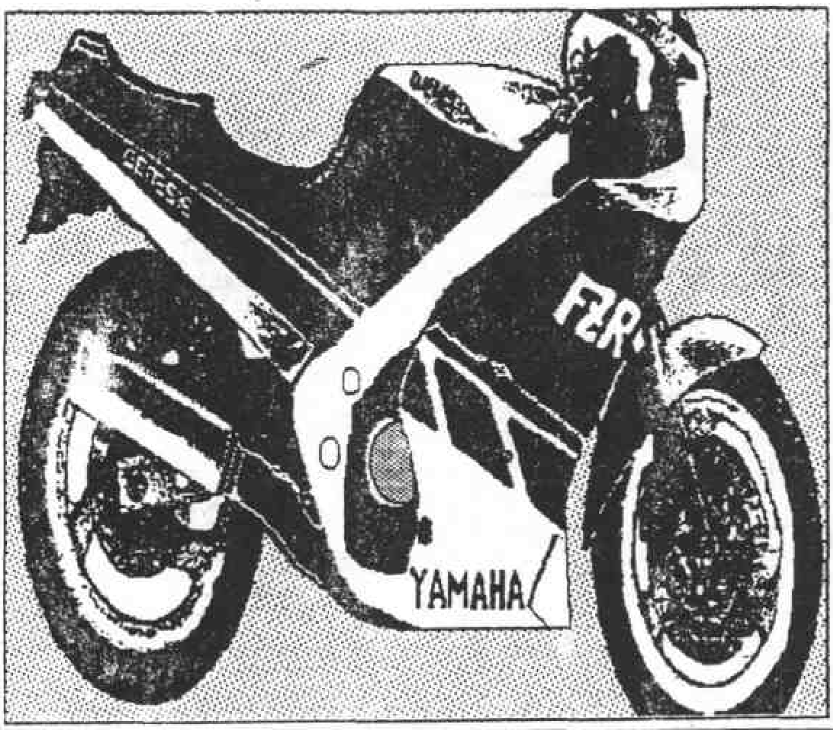
(1)



(2)



(3)



(4)

图 2-9　垂直交错程序 CROSSV.C 的显示效果

2.3.3 水平交错效果

水平交错效果为垂直交错效果的另一种表现方法，同样是将一张图分成奇、偶两部分，所不同的是本效果是将分成两组后的图形分别由左右两端移进屏幕。

水平交错和前两种效果最大的不同点在于本效果不再是以扫描线为搬移单位，而是以字节（在单色为 8 个像素）为单位。所以在搬移的过程中，必须再加入一个循环以控制所搬移的位数。

以下是水平交错效果的示意图：

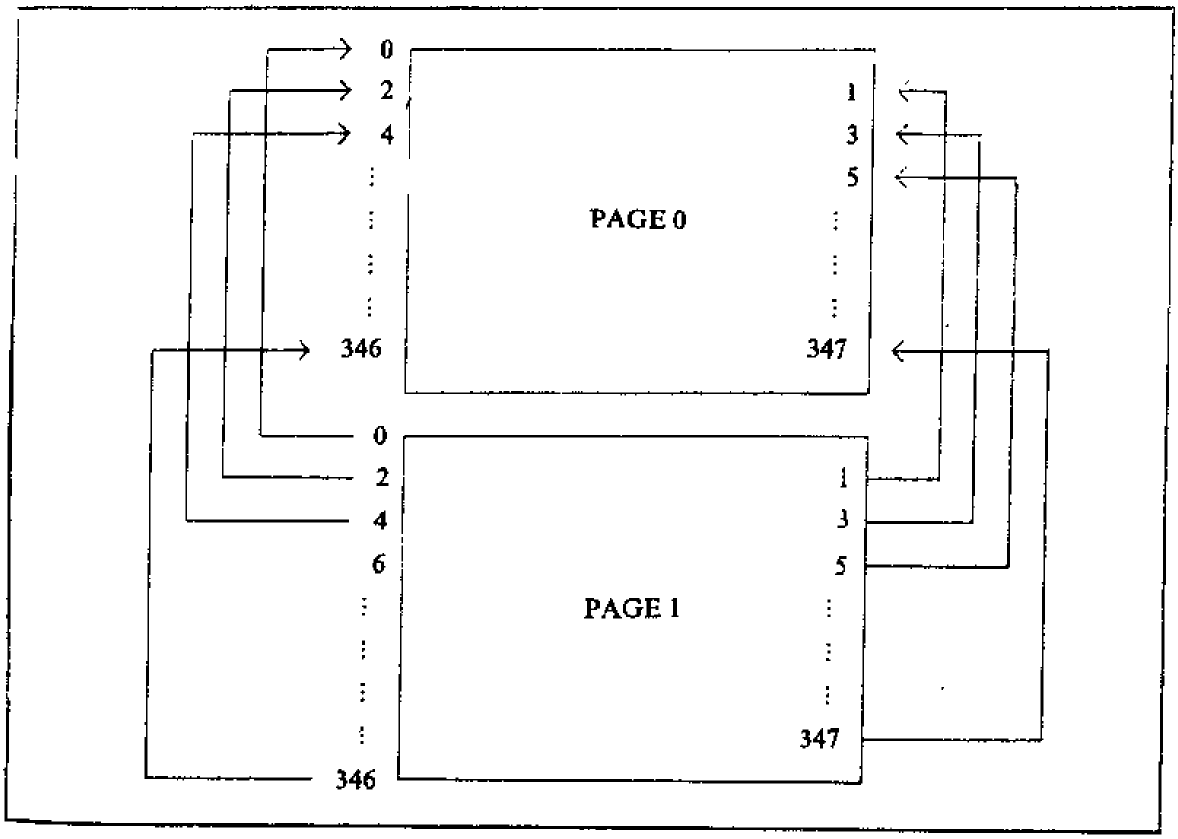


图 2-10　水平交错效果示意图

接着我们来看看水平交错效果的算法及其相对应的程序：

算法

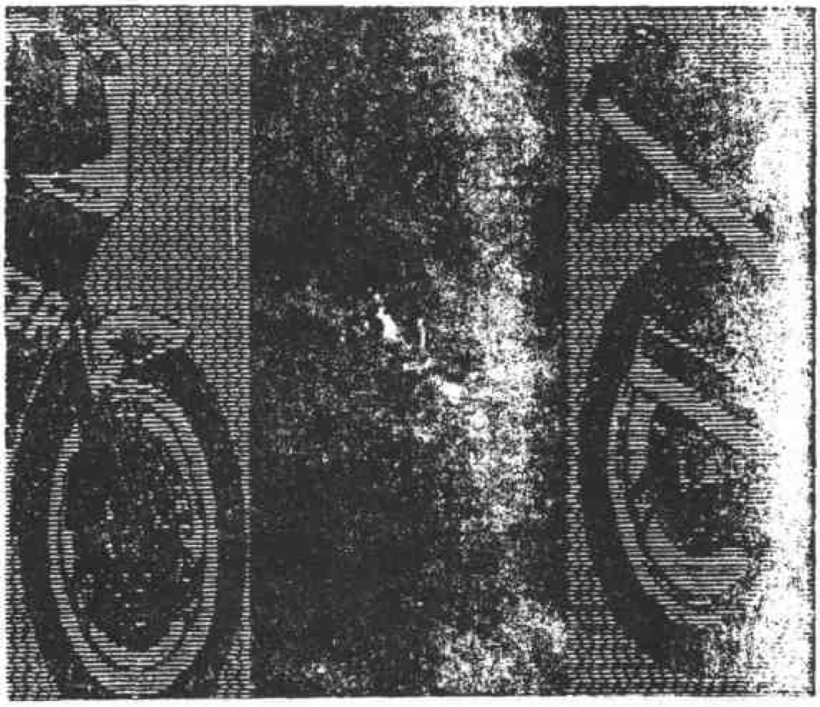
```
FOR I=1 TO 90
   FOR J=0 TO 347 STEP 2
      MOVE I BYTES FROM Page1(J)+90 -I TO Page0(J)
      MOVE I BYTES FROM Page1(J+1) T) Page(J+1)+90-I
   NEXT J
NEXT I
```

水平交错效果显示程序 CROSSH.C：

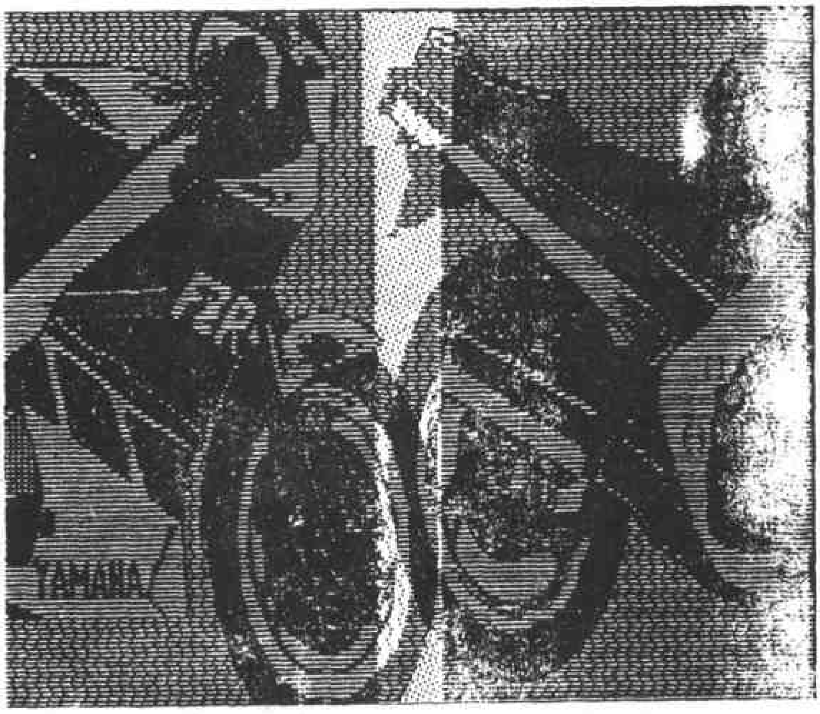
```
/*  CROSSH.C  */
```

```
#include <graphics.h>
#include <stdio.h>
#include <alloc.h>

void loadmapm(char * );
void map2(void);
void crooss_h2(void);

unsigned addr[348];

main()
{
      int graph_driver=HERCMONO, graph_mode=HERCMONOHI;

      initgraph(&graph_driver, &graph_mode, "");
      loadmapm("demo.map");   /* Load a picture into video page 1   */
      map2();                 /* Tabulate */
      cross_h2();             /* Show the picture using "cross_h" effect */
      getch();                /* Wait for pressing key state */
      closegraph();           /* Close graphics mode */
}

void loadmapm(char *name)
{
      FILE  *fp;
      char *buf;
      char low, high;
      unsigned size, x, y;
      union {
           char ch[2];
           int no;
      } value;

      if ((fp=fopen(name,"rb")) == NULL)    {
          puts("\a");
          closegraph();
          printf("\nFile reading error!!");
          exit(1);
      }
      else    {
          if ((unsigned)(filelength(fileno(fp))) >= coreleft())
          {
              puts("\a");
```

```
                closegraph();
                printf ("\nSorry! Memory is not enough.\n");
                exit(1);
            }
            else
                size = filelength(fileno(fp));
            buf = (char * ) malloc(size);
            setvisualpage(0);
            setactivepage(1);
            fread(buf,1,size,fp);
            value.ch[0] = buf[7]
            value.ch[1] = buf[8];
            x = value.no;
            value.ch[0] = buf[9];
            value.ch[1] = buf[10];
            y = value.no;
            x = (720-x) / 2;
            y = (348-y) / 2;
            clearviewport();
            buf[7]--; buf[9]--;
            putimage(x,y,buf+7,COPY_PUT);
            fclose(fp);
            free(buf);
        }
}

void map2()
{
            register int i, j;

            for(i=0; i<348; i++)
                addr[i] = 8192 * (i%4) + 90 * (i / 4);
}

void cross_h2()
{
      int i, j;

      for(i=1; i<=90; i++)
      for(j=0; j<348; j+=2)        {
            movedata(0xb800, addr[j]+90-i, 0xb000, addr[j], i);
            movedata(0xb800, addr[j+1], 0xb000, addr[j+1]+90-i, i);
      }
}
```



(1)



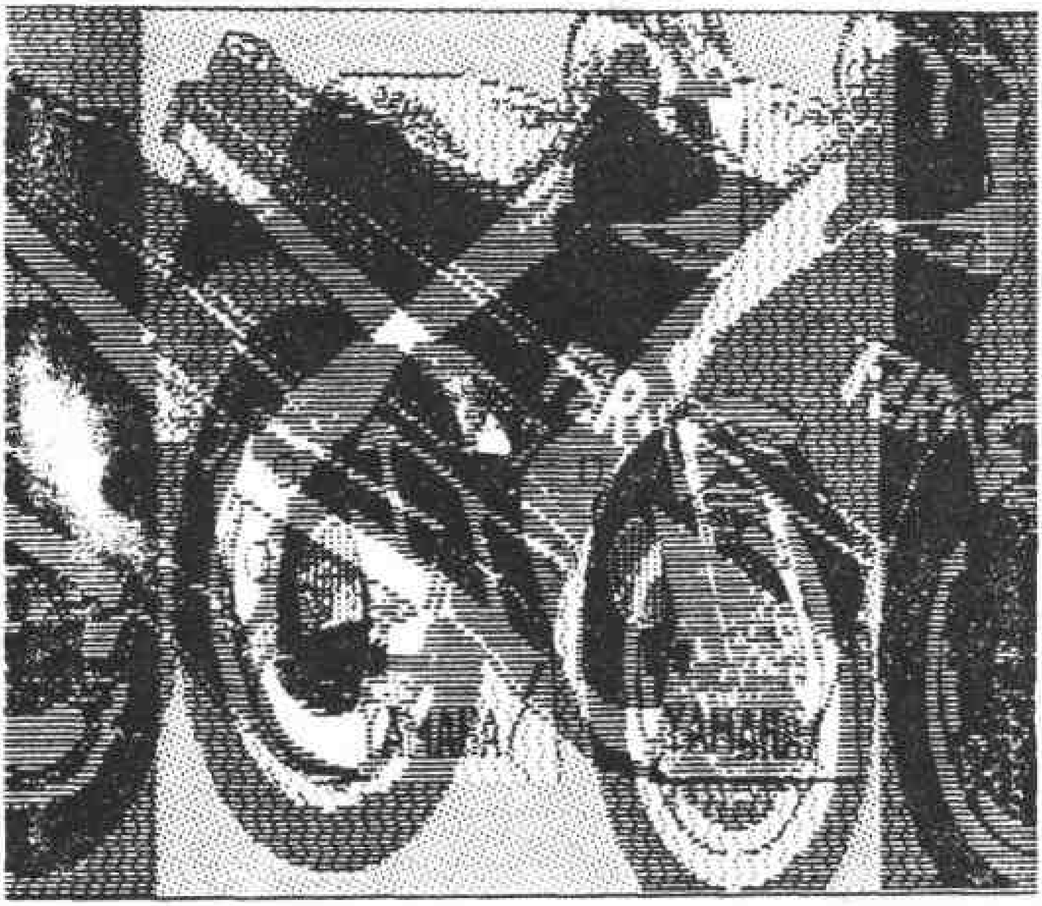
(2)

雨状效果是在屏幕上看到图形数据由上而下流动，而将图形由下而上逐渐堆积成完整的图形．由于它在显示的过程中具有流动的效果，因此有人又称之为流水效果或流动效果．

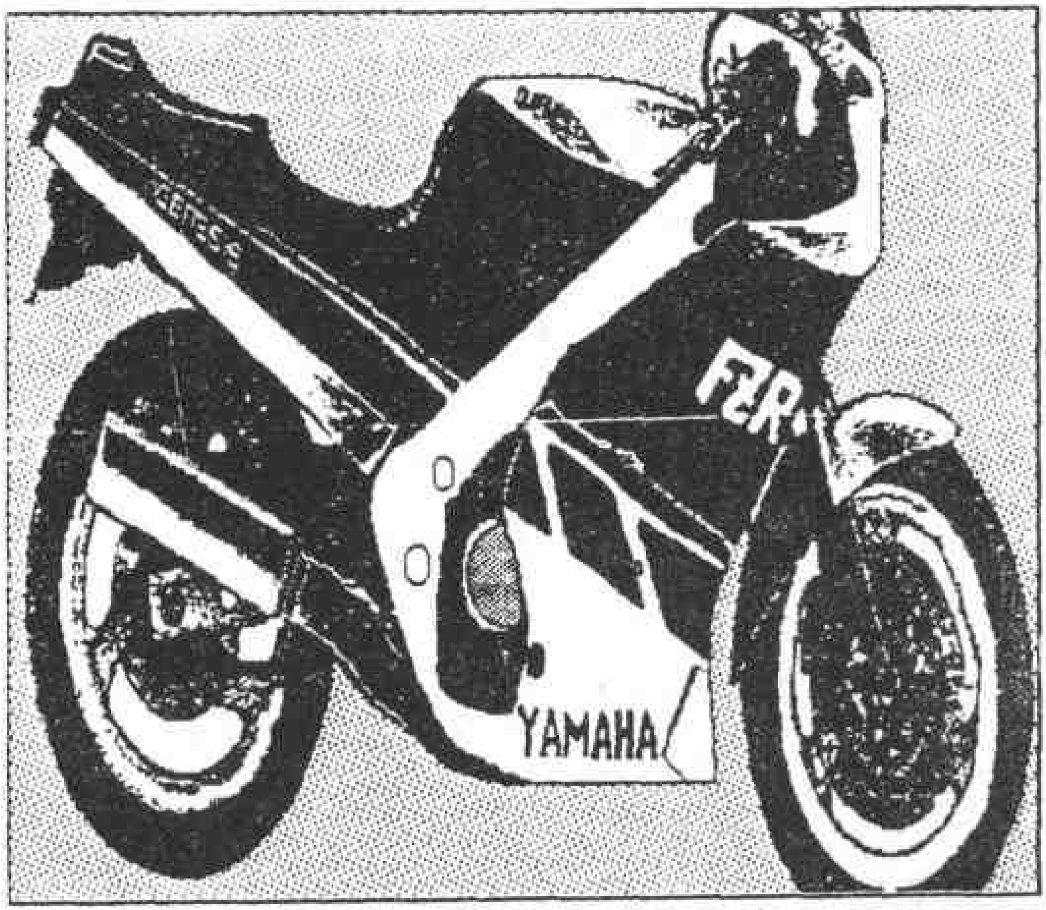
图 2-11　水平交错程序 CROSSH.C 的执行结果

2.3.4　雨状效果

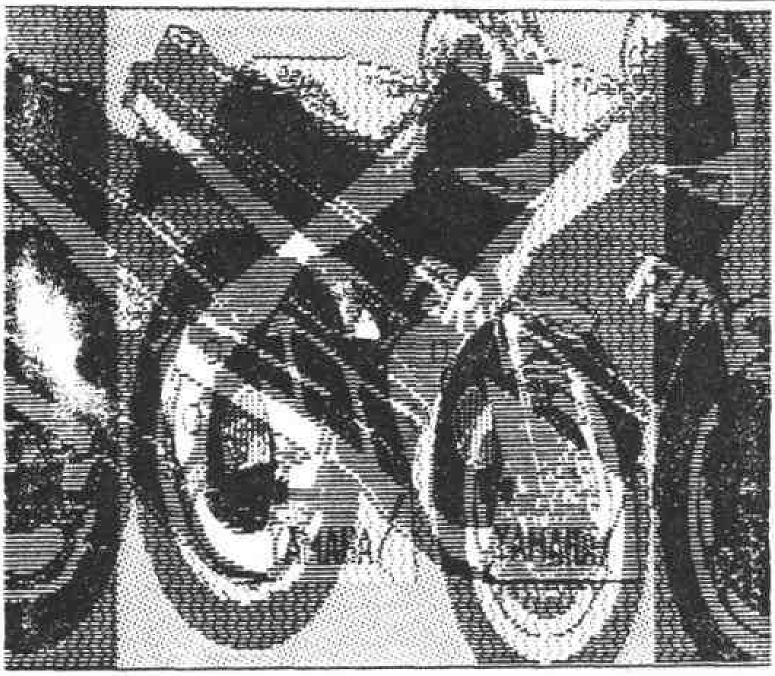
雨状效果是在屏幕上看到图形数据由上而下流动，而将图形由下而上逐渐堆积成完整的图形．由于它在显示的过程中具有流动的效果，因此有人又称之为流水效果或流动效果．



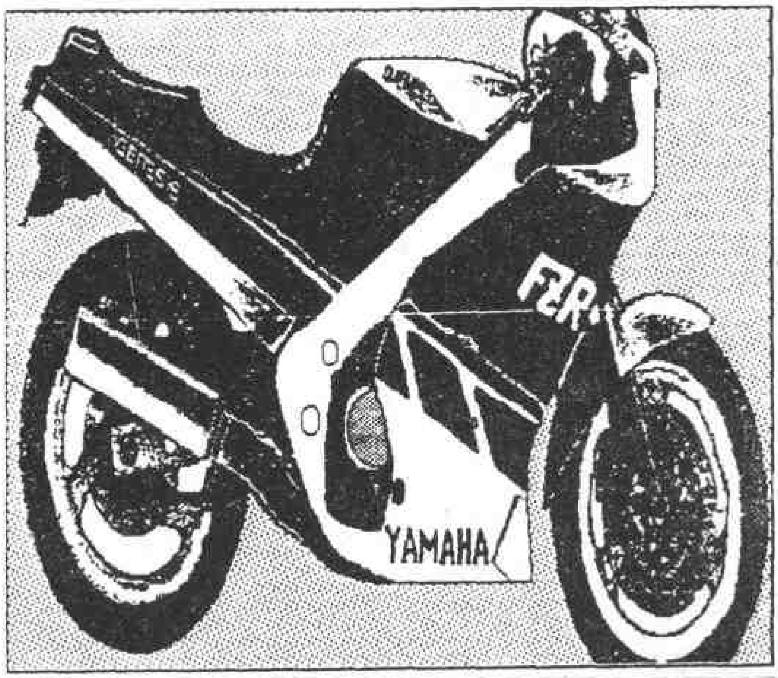
(3)



(4)



(3)



(4)

图 2-11　水平交错程序 CROSSH.C 的执行结果

2.3.4　雨状效果

雨状效果是在屏幕上看到图形数据由上而下流动，而将图形由下而上逐渐堆积成完整的图形。由于它在显示的过程中具有流动的效果，因此有人又称之为流水效果或流动效果。

本效果的原理为:

1. 先从暂存页的图形取得最后一条扫描线（扫描线347），然后将这条扫描线的图形数据，依序搬移可视页的第一条扫描线（扫描线 0）到最后一条扫描线（扫描线 347），让此条扫描线在屏幕上留下它的轨迹。
2. 接着再从暂存页取得倒数第二条（扫描线346），然后将这条扫描线的图形数据，依序搬移至可视页的第一条扫描线到倒数第二条扫描线（扫描线 346）。
3. 其余的扫描线依此类推。

因为每一条扫描线皆会在屏幕上留下轨迹，然后再移至定位，雨状的效果便因此而达到。本效果有多种变化：向上、向下、向左、向右，留给读者自行变化。以下是本效果的示意图:

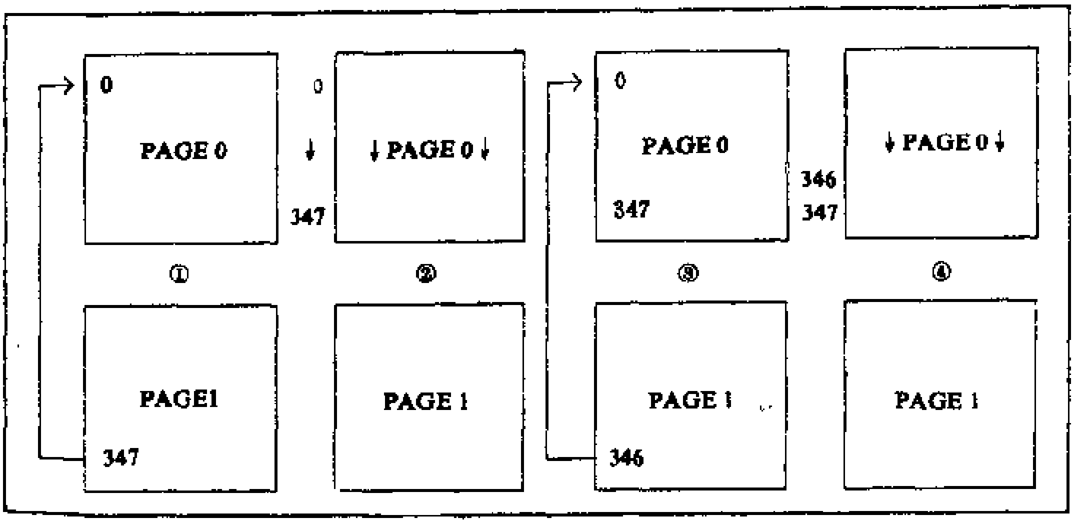


图 2-12 雨状效果示意图

下面是雨状效果的算法及其相应的程序:

算法

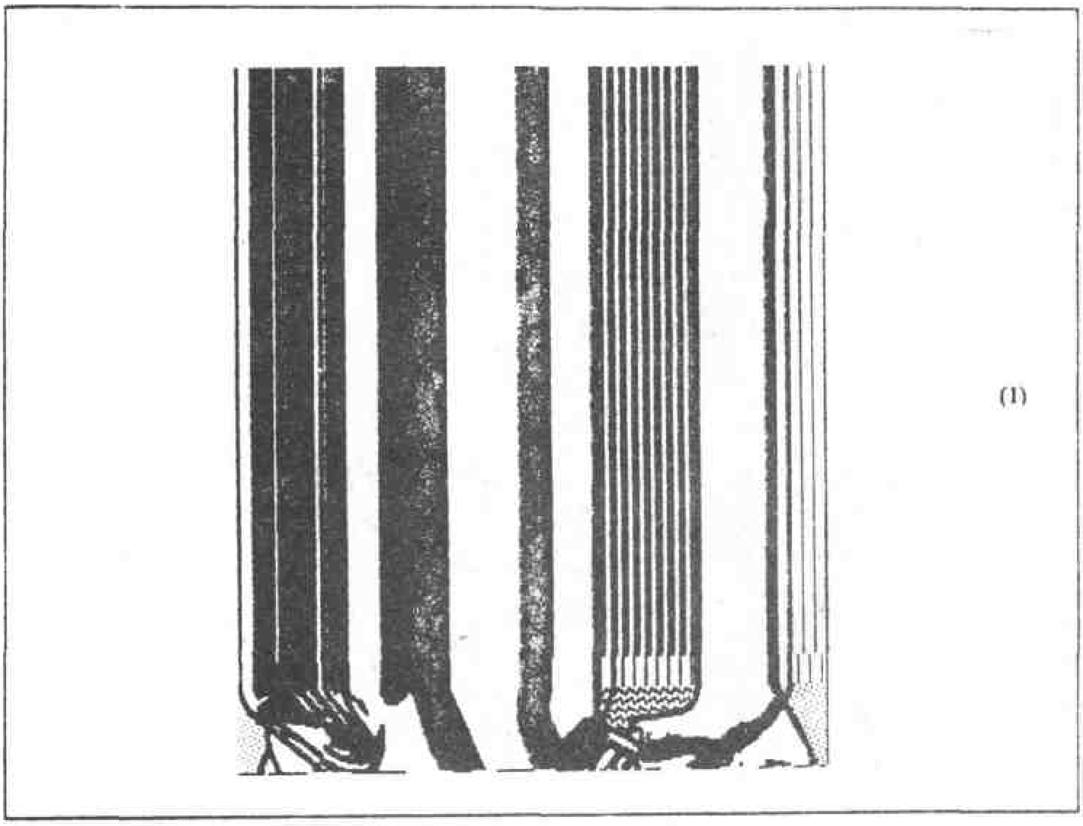
```
FOR I=347 TO 0 STEP -1
  FOR J=0 TO I
   MOVELINE Page1(I) TO Page0(J)
   NEXT J
NEXT I
```

雨状效果显示程序 RAIN.C

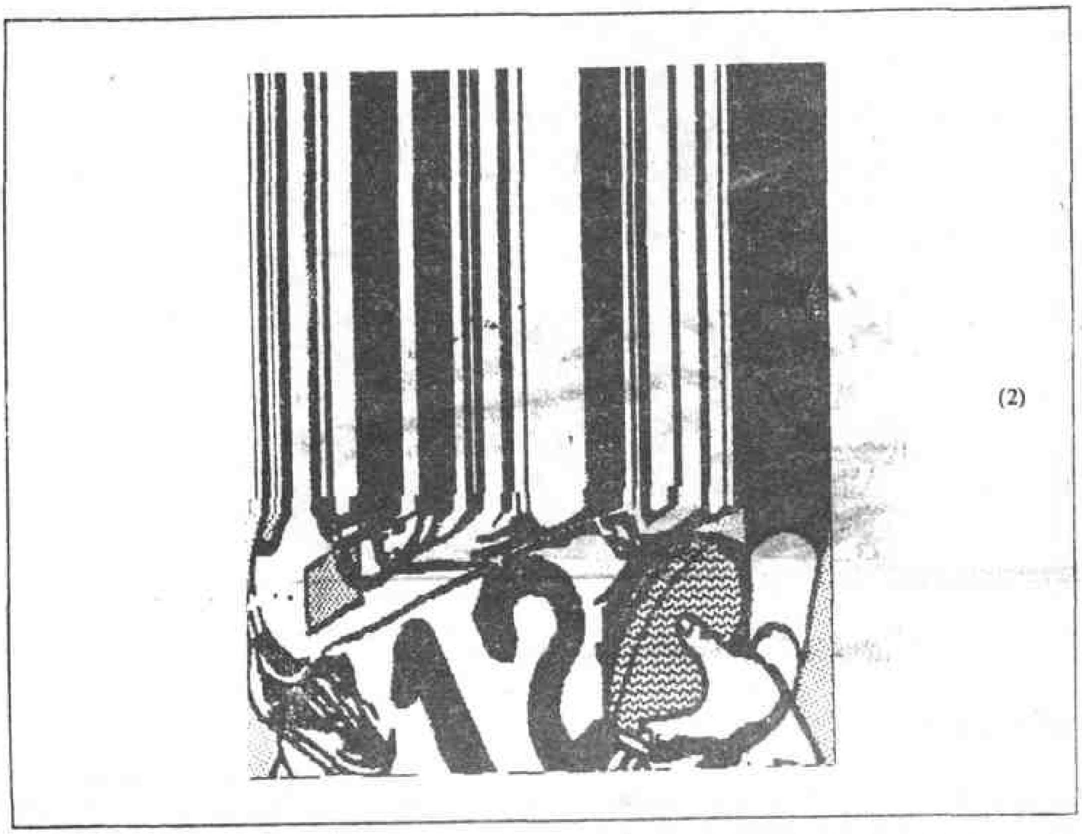
```
/ * RAIN.C * /

#include <graphics.h>
#include <stdio.h>
#include <alloc.h>

void loadmapm(char *);
```

```
void map2(void);
void rain2(void);

unsigned addr[348];

main()
{
    int graph_driver=HERCMONO, graph_mode=HERCMONOHI;

    initgraph(&graph_driver, &graph_mode, "");
    loadmapm("demo.map");    /* Load a picture into video page 1  */
    map2();                  /* Tabulate */
    rain2();                 /* Show the picture using "rain" effect */
    getch();                 /* Wait for pressing key state */
    closegraph();            /* Close graphics mode */
}

void loadmapm(char *name)
{
    FILE *fp;
    char *buf;
    char low, high;
    unsigned size, x, y;
    union {
        char ch[2];
        int no;
    } value;

    if ((fp=fopen(name,"rb")) == NULL)    {
        puts("\a");
        closegraph();
        printf("\nFile reading error!!");
        exit(1);
    }
    else    {
        if ((unsigned)(filelength(fileno(fp))) >= coreleft())
        {
            puts("\a");
            closegraph();
            printf ("\nSorry! Memory is not enough.\n");
            exit(1);
        }
    else
        size=filelength(fileno(fp));
```

```
        buf=(char  * ) malloc(size);
        setvisualpage(0);
        setactivepage(1);
        fread(buf,1,size,fp);
        value.ch[0] = buf[7];
        value.ch[1] = buf[8];
        x = value.no;
        value.ch[0] = buf[9];
        value.ch[1] = buf[10];
        y = value.no;
        x = (720-x)/2;
        y = (348-y)/2;

        clearviewport();
        buf[7]--; buf[9]--;
        putimage(x,y,buf+7,COPY_PUT);
        fclose(fp);
        free(buf);
      }
}

void map2()
{
      register int i, j;

      for(i=0; i<348; i++)
          addr[i] = 8192*(i%4) + 90*(i/4);
}

void rain2()
{
      int i, j;

      for(i=347;i>=0;i--)
      for(j=0;j<=i;j++)
          movedata(0xb800, addr[i], 0xb000, addr[j], 90);
}
```



(1)

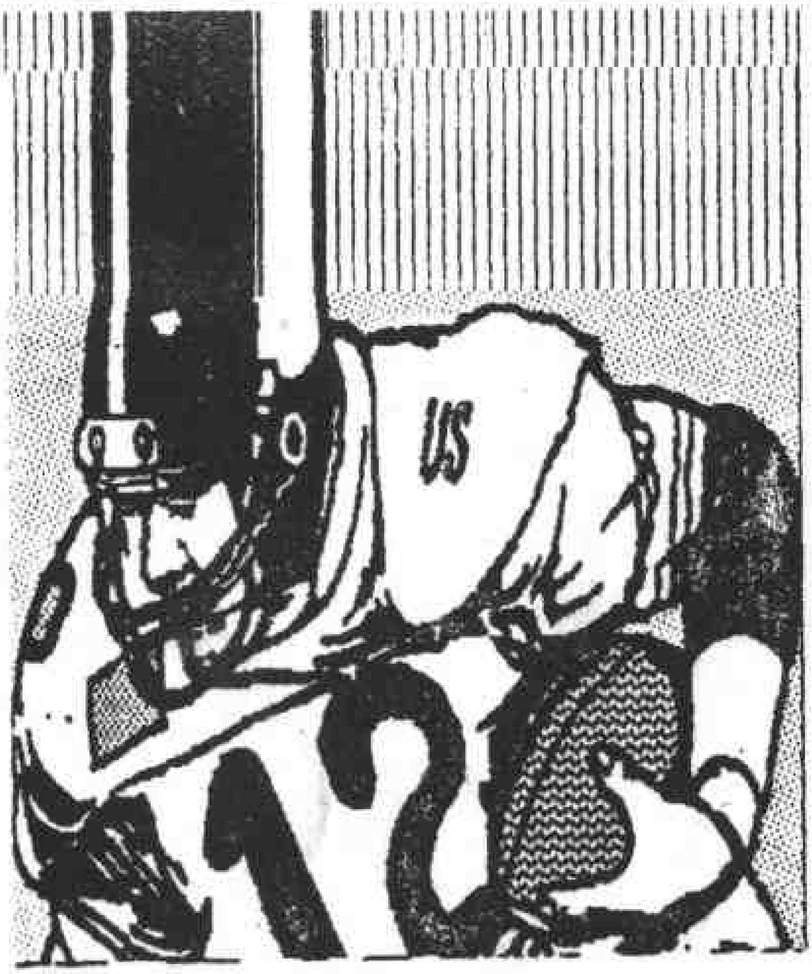


(2)

图 2-13　雨状效果程序 RAIN.C 的执行结果

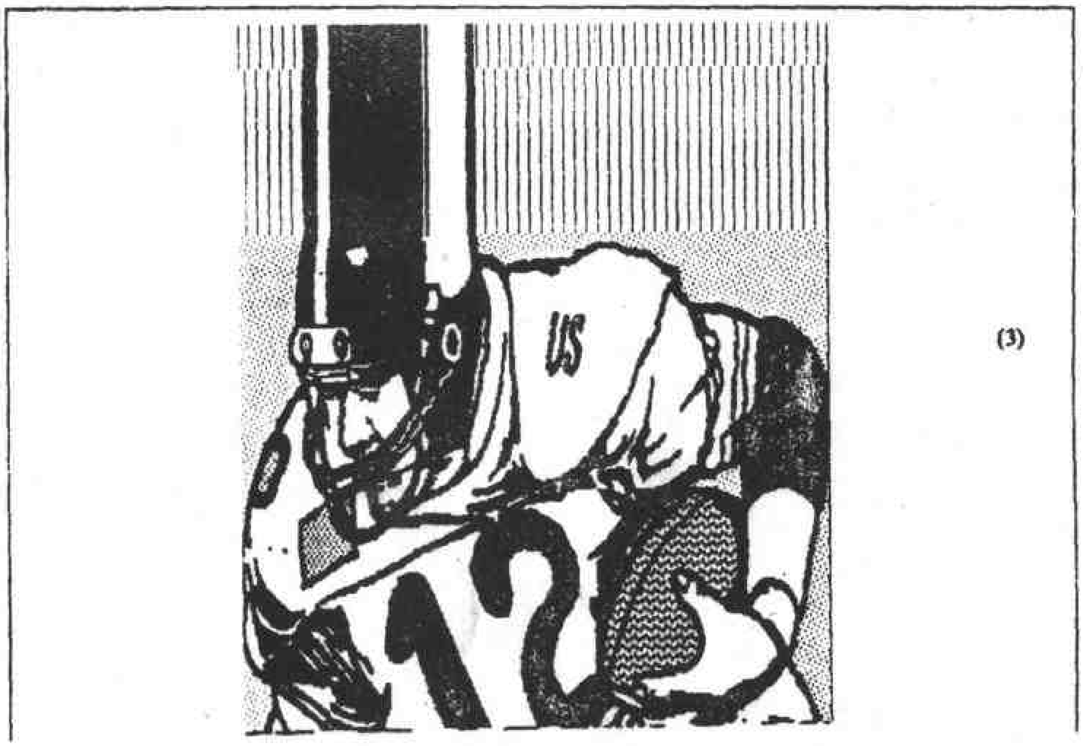
2.3.5　百叶窗效果

在日常生活中，百叶窗是许多人家中不可缺少的遮阳窗饰之一。本效果即是模仿当你将百叶窗叶子打开，逐渐看到窗外景色的过程，并将此过程显示在屏幕上，

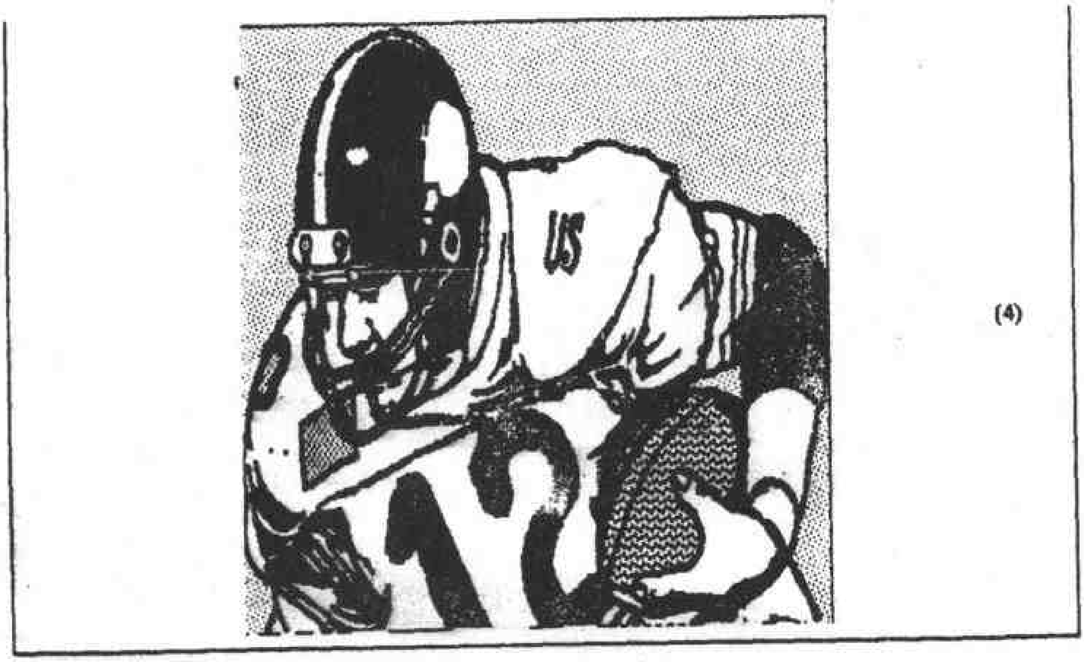


(3)

(4)



(3)



(4)

图 2-13　雨状效果程序 RAIN.C 的执行结果

2.3.5　百叶窗效果

在日常生活中，百叶窗是许多人家中不可缺少的遮阳窗饰之一。本效果即是模仿当你将百叶窗叶子打开，逐渐看到窗外景色的过程，并将此过程显示在屏幕上。

本效果的原理是将放在暂存页中的图形数据分成若干组（以十组左右为宜），然后依次从第一组到最后一组搬移，第一次每组各搬移第一条扫描线至可视页的相对应位置，第二次搬移第二条扫描线，接着为第三条绘图，第四条扫描线……直到将各组之图形数据搬完为止。同样地，本效果也在向上、向下、向左、向右等四种变化。以下是本效果的示意图：

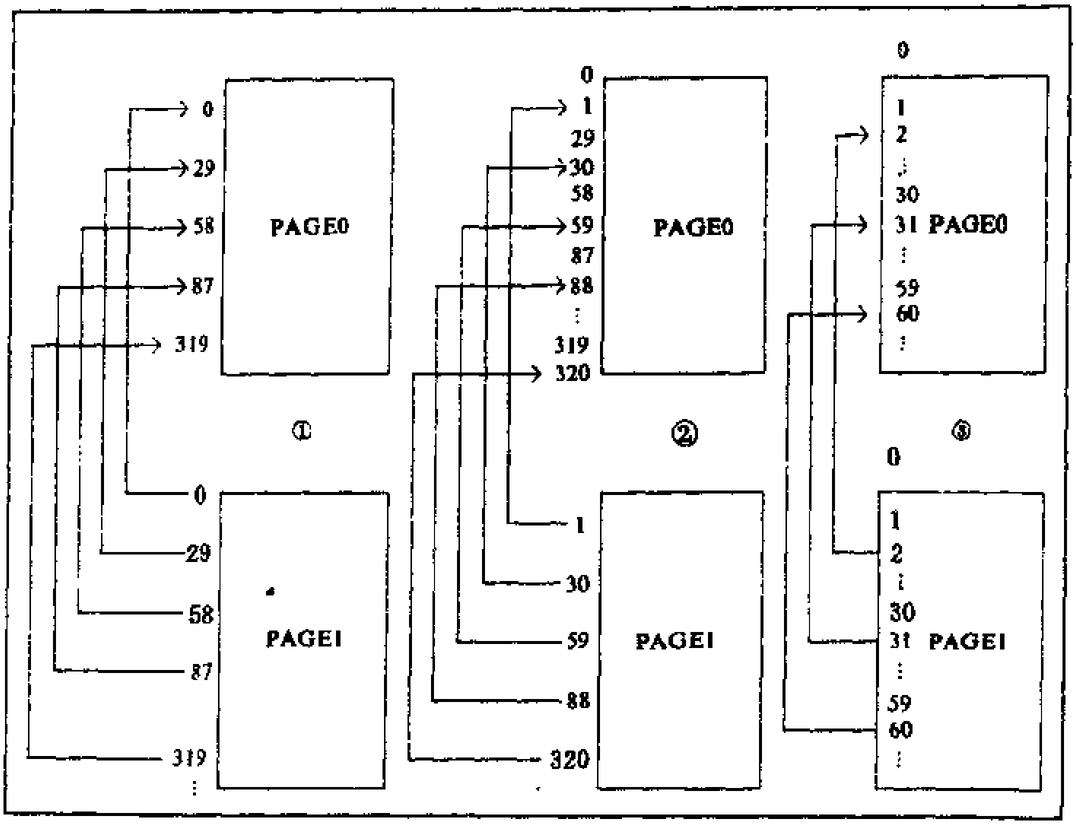


图 2-14 百叶窗效果示意图

接下来的算法是以单色屏幕为讨论对象，因为单色 Hercules 适配器共有 348 条扫描线，能为 12 整除，故将它分成 12 组，每组 29 条扫描线：

算法

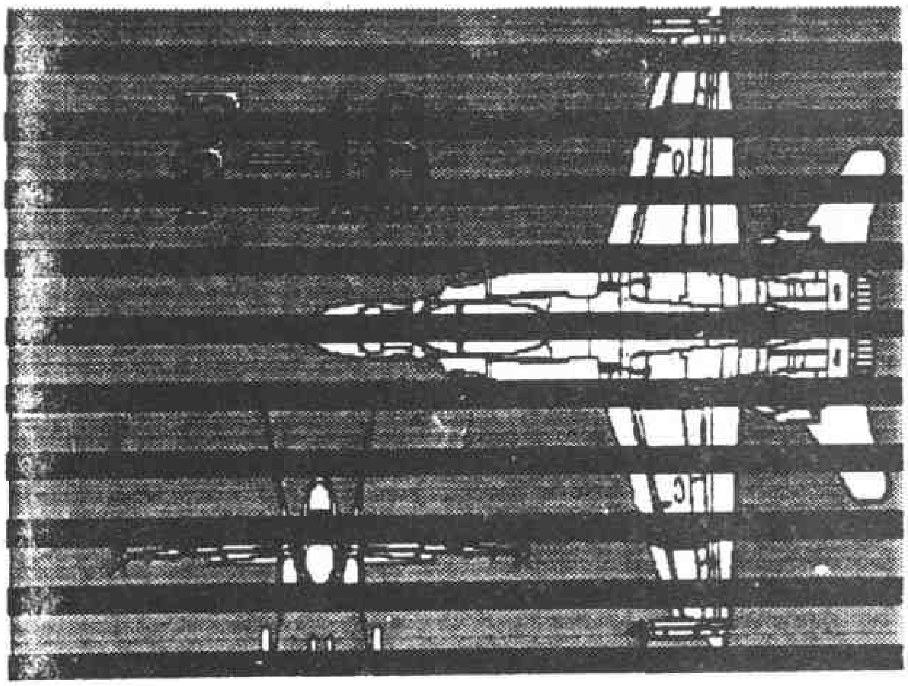
```
FOR I=0 TO 28
  FOR J=0 TO 11
   MOVELINE Page1(29*J+I) TO Page0(29*J+I)
  NEXT J
NEXT I
```

百叶窗效果显示程序 CURTAIN.C

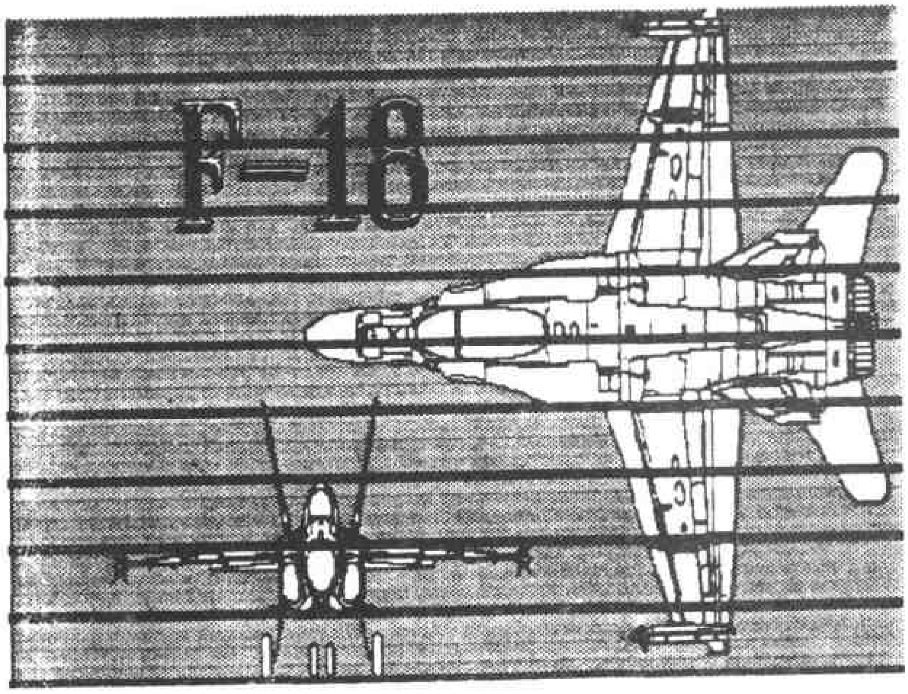
```
/*  CURTAIN.C  *
```

```
#include <graphics.h>
#include <stdio.h>
#include <alloc.h>

void loadmapm(char * );
void map2(void)
void curtain2(void);

unsigned addr[348];

main()
{
      int graph_driver=HERCMONO, graph_mode=HERCMONOHI;

      initgraph(&graph_driver, &graph_mode, ""); /* Set graphics mode */
      loadmapm("demo.map");  /* Load a picture into video page 1 */
      map2();                 /* Tabulate */
      curtain2();             /* Show the picture using "curtain" effect */
      getch();                /* Wait for pressing key state */
      closegraph();           /* Close graphics mode */
}

void loadmapm(char *name)
{
      FILE *fp;
      char *buf;
      char low, high;
      unsigned size, x, y;
      union {
           char ch[2];
           int no;
      } value;

      if ((fp=fopen(name,"rb")) == NULL)    {
         puts("\a");
         closegraph();
         printf("\nFile reading error!!");
         exit(1);
      }
      else     {
          if ((unsigned)(filelength(fileno(fp))) >= coreleft())
          {
               puts("\a");
               closegraph();
```

```
            printf ("\nSorry! Memory is not enough.\n");
            exit(1);
        }
        else
            size = filelength(fileno(fp));
        buf = (char *) malloc(size);
        setvisualpage(0);
        setactivepage(1);
        fread(buf,1,size,fp);
        value.ch[0] = buf[7];
        value.ch[1] = buf[8];
        x = value.no;
        value.ch[0] = buf[9];
        value.ch[1] = buf[10];
        y = value.no;
        x = (720-x) / 2;
        y = (348-y) / 2;

        clearviewport();
        buf[7]--; buf[9]--;
        putimage(x,y,buf+7,COPY_PUT);
        fclose(fp);
        free(buf);
    }
}

void map2()
{
    register int i, j;

    for(i=0; i<348; i++)
    addr[i] = 8192*(i%4) + 90*(i/4);
}

void curtain2()
{
    int i, j;

    for(i=0;i<=28;i++)     {
        delay(30);
    for(j=0;j<=11;j++)
        movedata(0xb800,addr[29*j+i],0xb000,addr[29*j+i],90);
    }
}
```



(1)



(2)

图 2-15 百叶窗效果程序 CURTAIN.C 的执行结果

2.3.6 积木效果

积木效果顾名思义，是模仿积木，将要显示的图形分成几个部分，然后一层一层在屏幕上堆积成原图形所产生的视觉效果。本效果究其原理是前两节雨状效果的一种变化，所不同的是积木效果每次所搬移的图形数据并不仅是一条扫描线，而是一块图形（即数条线一起搬移），而且在屏幕上移动时并不留下轨迹。为了达到本效果，我们做了如下的处理：

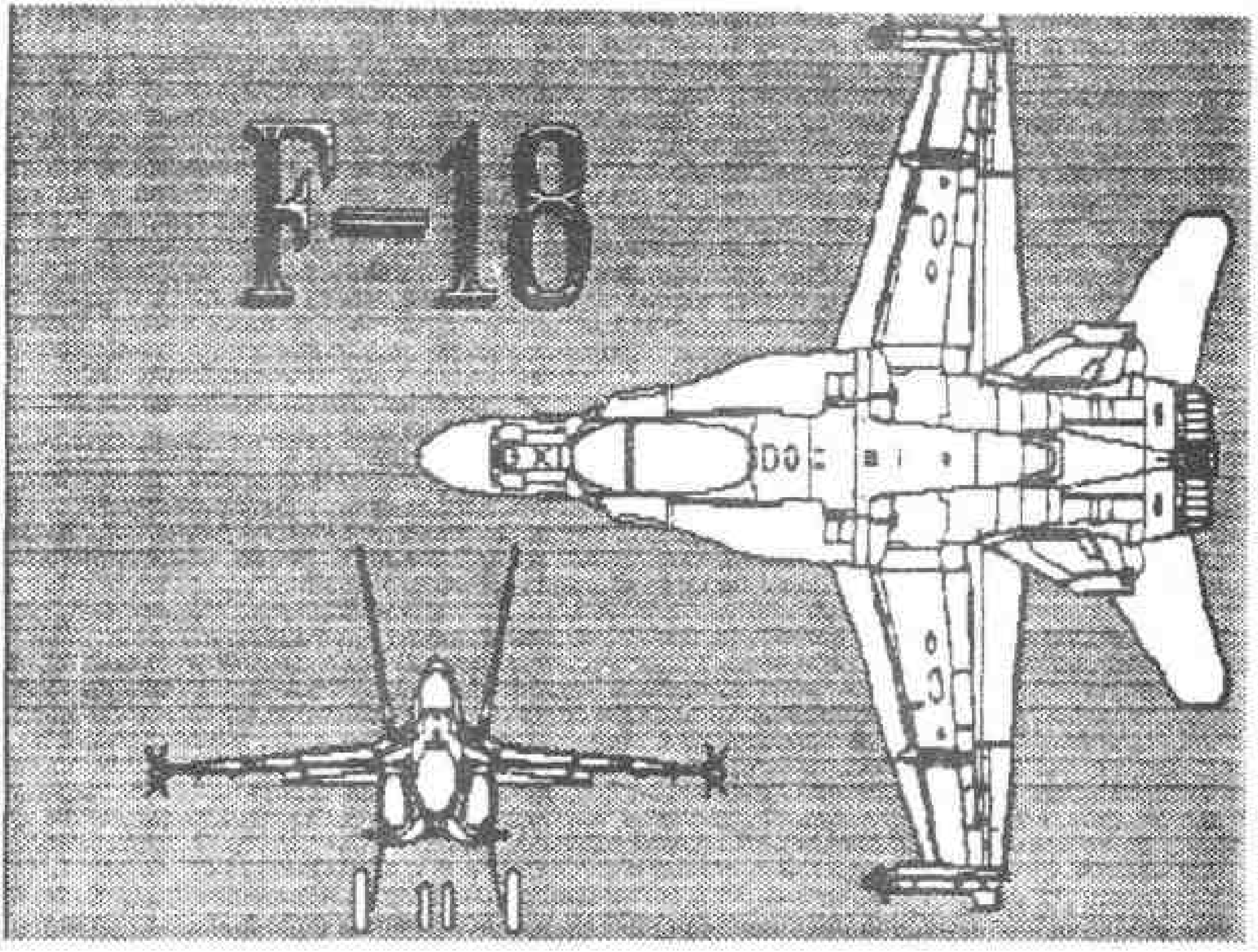
1. 每次搬移 4 条扫描线的图形数据。
2. 在屏幕上移动时，为了避免留下轨迹，在移动一条扫描线时，立即消除不需要的图形数据（填充空白）。

积木效果的算法如下：

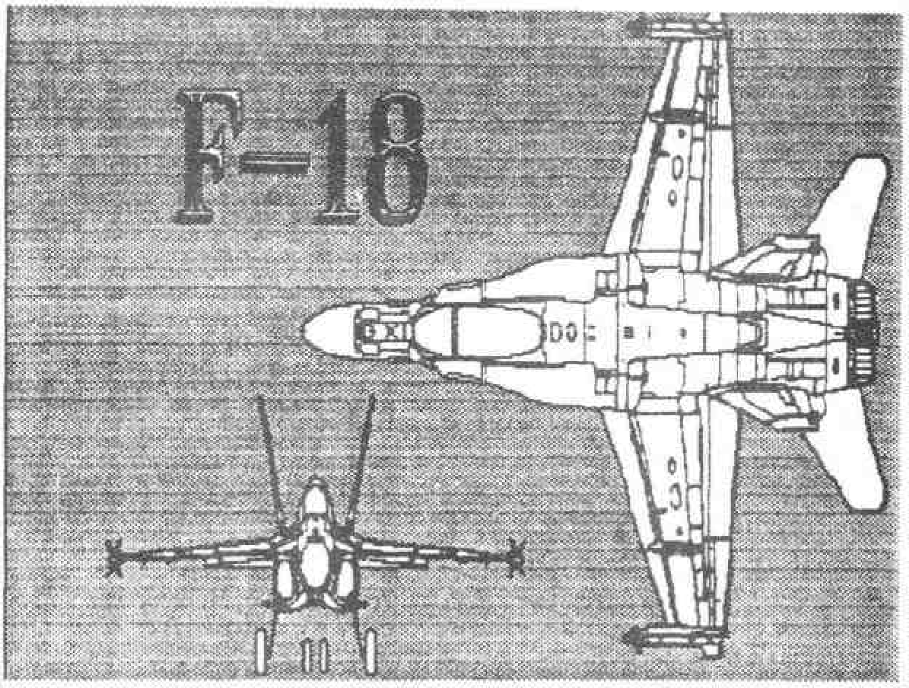
```
算法
FOR I=344 TO 0 STEP -4
  FOR J TO I
    FOR K=0 TO 3
     MOVELINE Page1(I+K) TO Page0(J+K)
    NEXT K
    ERASELINE Page0(J+K)
  NEXT J
NEXT I
```

其中 ERASELINE Page0(J-1)表示将第 0 页的扫描线 J-1 之图形数据存放区全部填入空白（0）。接下来是 C 程序：

```
/ *  BLOCK.C  * /
```



(3)



(3)

图 2-15 百叶窗效果程序 CURTAIN.C 的执行结果

2.3.6 积木效果

积木效果顾名思义，是模仿积木，将要显示的图形分成几个部分，然后一层一层在屏幕上堆积成原图形所产生的视觉效果。本效果究其原理是前两节雨状效果的一种变化，所不同的是积木效果每次所搬移的图形数据并不仅是一条扫描线，而是一块图形（即数条线一起搬移），而且在屏幕上移动时并不留下轨迹。为了达到本效果，我们做了如下的处理：

1. 每次搬移 4 条扫描线的图形数据。
2. 在屏幕上移动时，为了避免留下轨迹，在移动一条扫描线时，立即消除不需要的图形数据（填充空白）。

积木效果的算法如下：

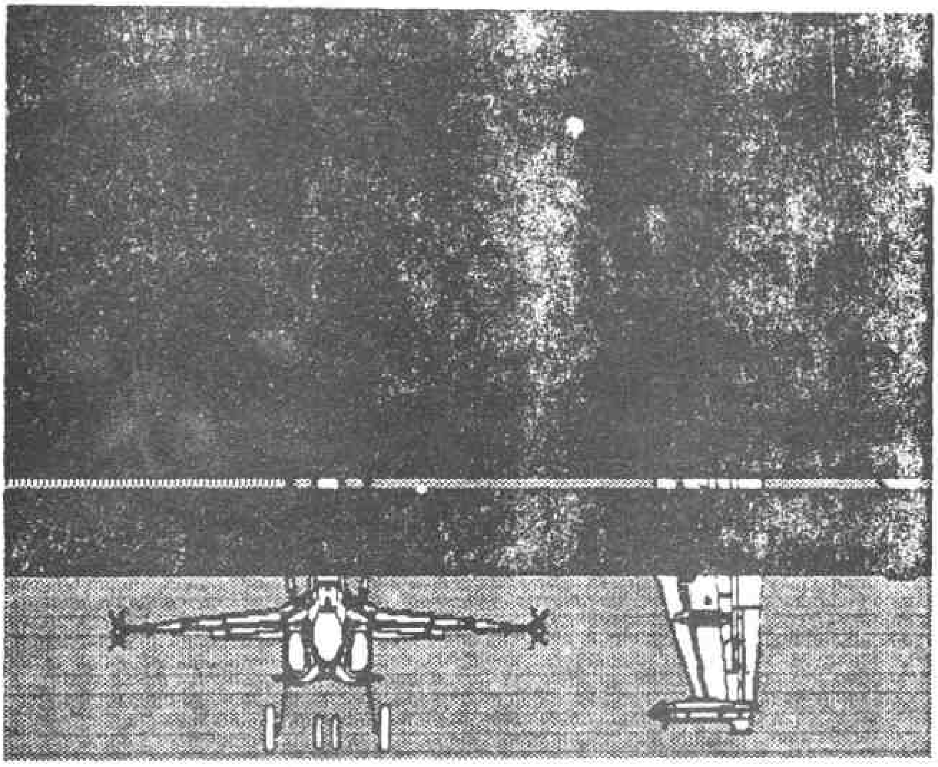
```
算法
FOR I=344 TO 0 STEP -4
  FOR J TO I
    FOR K=0 TO 3
     MOVELINE Page1(I+K) TO Page0(J+K)
    NEXT K
    ERASELINE Page0(J+K)
  NEXT J
NEXT I
```

其中 ERASELINE Page0(J-1)表示将第 0 页的扫描线 J-1 之图形数据存放区全部填入空白（0）。接下来是 C 程序：

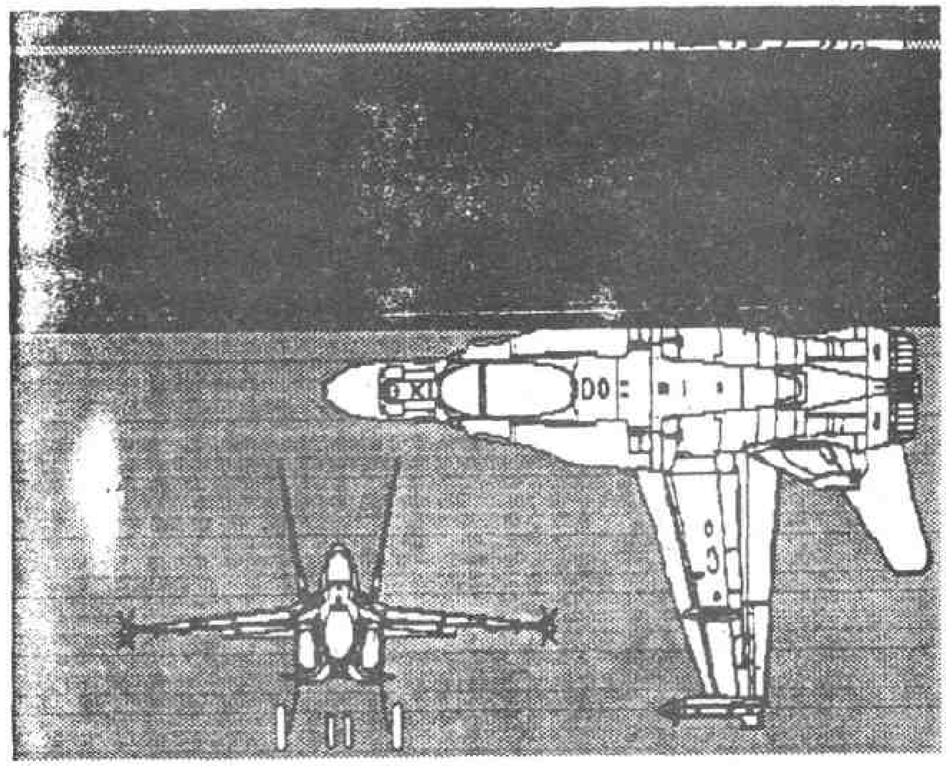
```
/*  BLOCK.C  */
```

```
#include <graphics.h>
#include <stdio.h>
#include <alloc.h>

void loadmapm(char *);
void map2(void);
void block2(void);

unsigned addr[348];

main()
{
      int graph_driver=HERCMONO, graph_mode=HERCMONOHI;

      initgraph(&graph_driver, &graph_mode, "");
      loadmapm("demo.map");    /* Load a picture into video page 1  */
      map2();                  /* Tabulate */
      block2();                /* Show the picture using "block" effect */
      getch();                 /* Wait for pressing key state */
      closegraph();            /* Close graphics mode */
}

void loadmapm(char *name)
{
     FILE  *fp;
     char  *buf;
     char low, high;
     unsigned size, x, y;
     union {
          char ch[2];
          int no;
     } value;

     if ((fp=fopen(name,"rb")) == NULL)     {
         puts("\a");
         closegraph();
         printf("\nFile reading error!!");
         exit(1);
     }
     else     {
         if ((unsigned)(filelength(fileno(fp))) >= coreleft())
         {
            puts("\a");
            closegraph();
            printf ("\nSorry! Memory is not enough.\n");
```

```
            exit(1);
        }
        else
            size = filelength(fileno(fp));
        buf = (char  * ) malloc(size);
        setvisualpage(0);
        setactivepage(1);
        fread(buf,1,size,fp);
        value.ch[0]  =  buf[7];
        value.ch[1]  =  buf[8];
        x  =  value.no;
        value.ch[0]  =  buf[9];
        value.ch[1]  =  buf[10];
        y  =  value.no;
        x  =  (720-x) / 2;
        y  =  (348-y) / 2;

        clearviewport();
        buf[7]--; buf[9]--;
        putimage(x,y,buf+7,COPY_PUT);
        fclose(fp);
        free(buf);
    }
}

void map2()
{
    register int i, j;

    for(i=0; i<348; i++)
        addr[i]  =  8192 * (i%4)  +  90 * (i / 4);
}

void block2()
{
    int i, j, k;

    for(i=344; i>=0; i-=4)
    for(j=1; j<=i; j++)        {
    for(k=0; k<4; k++)
        movedata(0xb800,addr[i+k],0xb000,addr[j+k],90);
    for(k=0; k<90; k++)
        pokeb(0xb000, addr[j-1]+k, 0);
    }
}
```



(1)



(2)

2.3.7 交替显示效果

在本章所谓的交替显示效果，是指两张图形同时出现在屏幕上，而其位置则做上下或者左右推移，类似于广告霓红灯的显示效果，如图 2–17 所示：

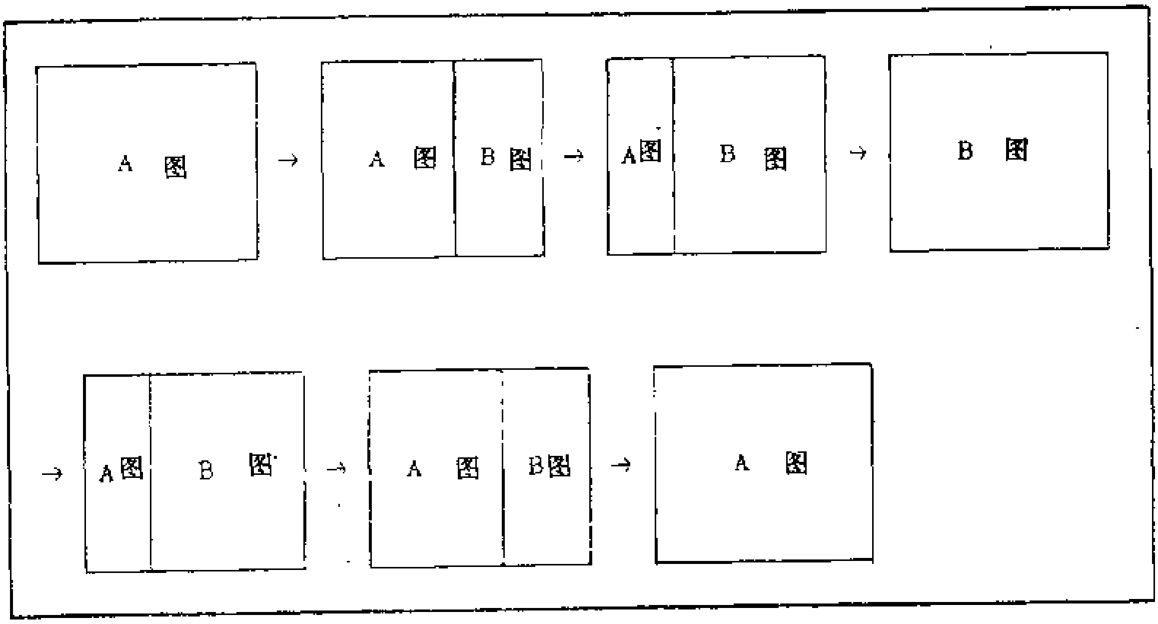
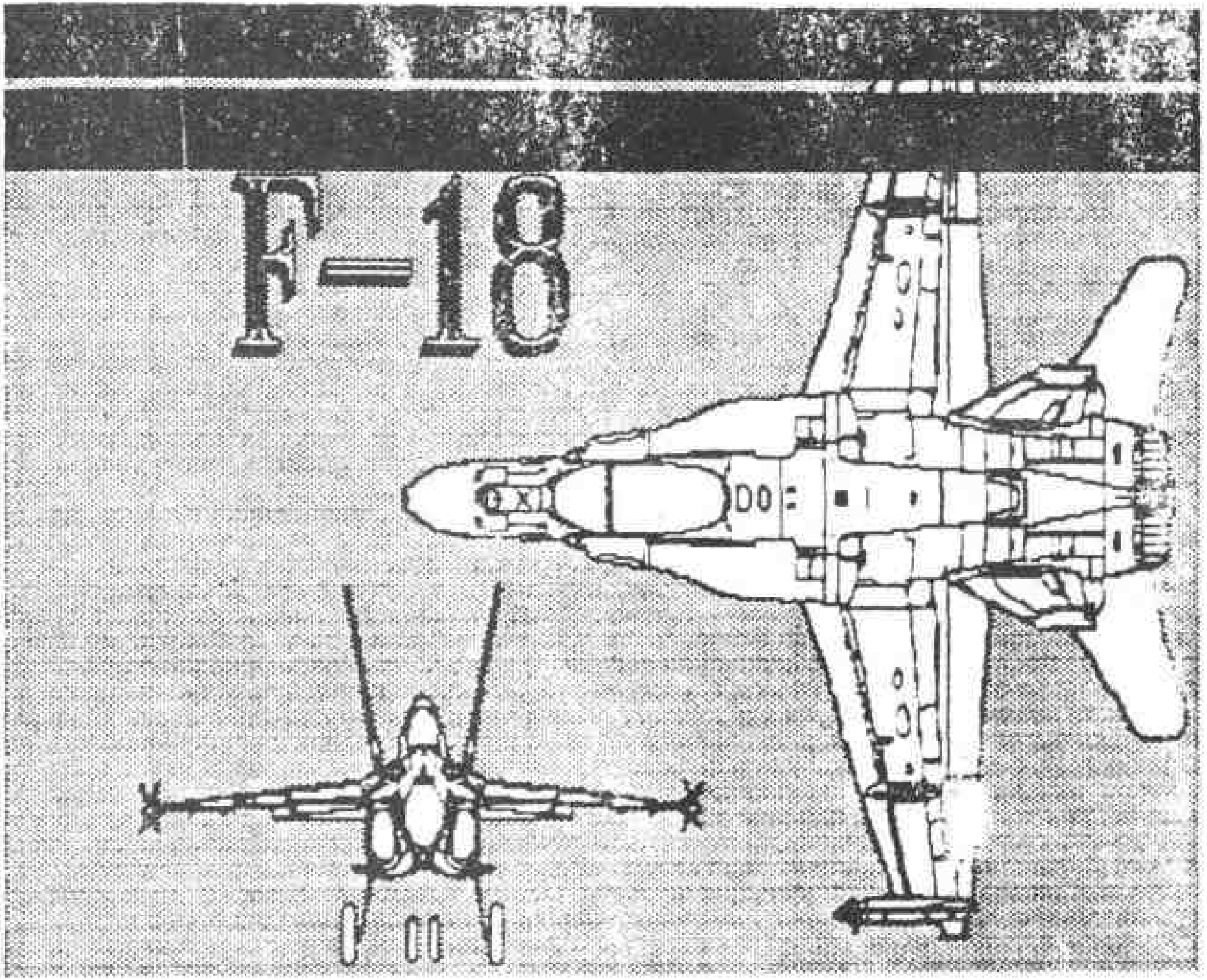


图 2–17 交替显示效果示意图



(3)

图 2-16 积木效果程序 BLOCK.C 的执行结果

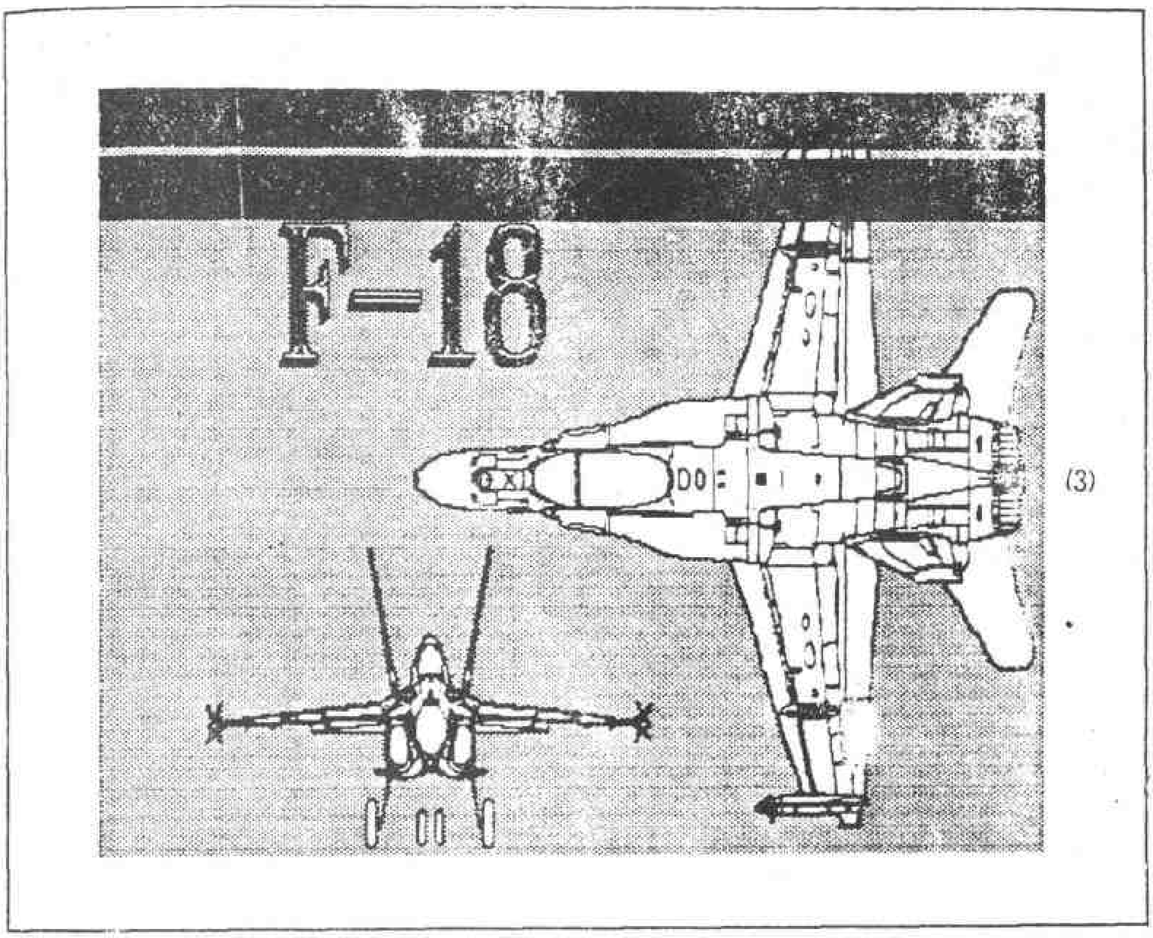


图 2-16 积木效果程序 BLOCK.C 的执行结果

2.3.7 交替显示效果

在本章所谓的交替显示效果，是指两张图形同时出现在屏幕上，而其位置则做上下或者左右推移，类似于广告霓红灯的显示效果，如图 2-17 所示：

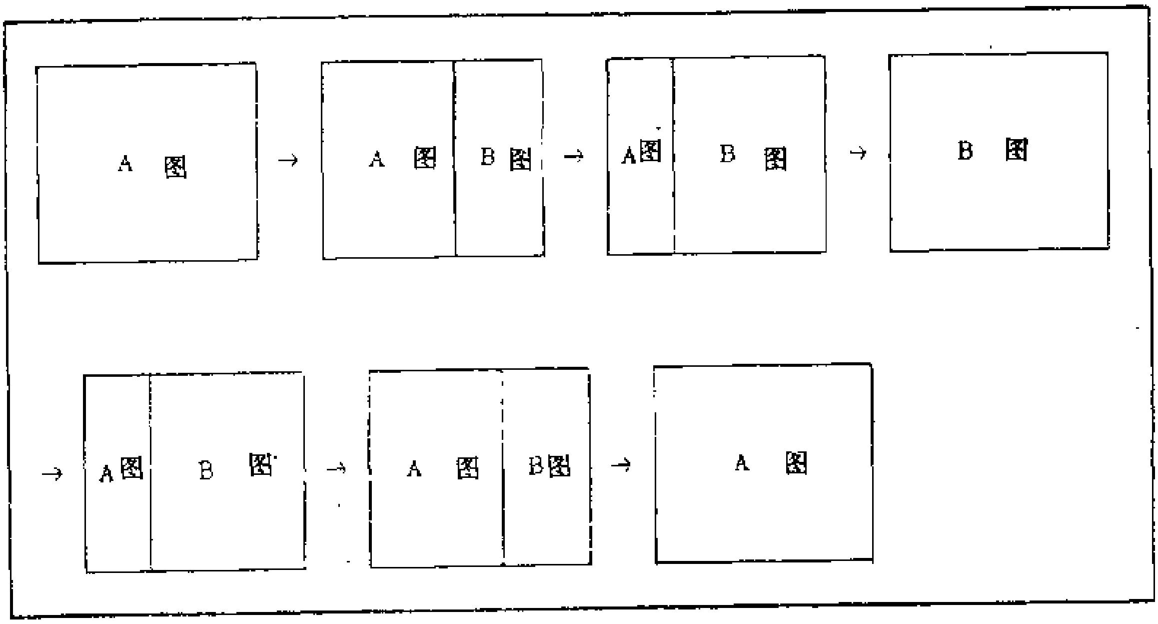
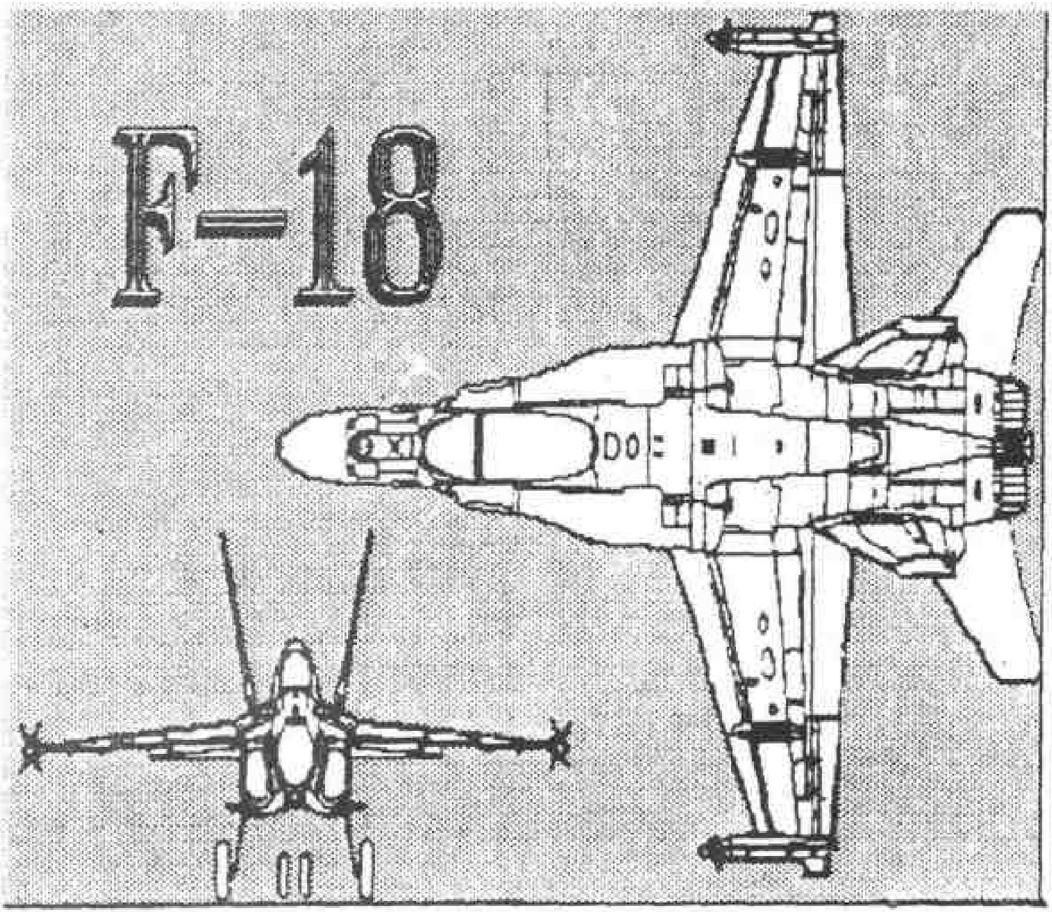


图 2-17 交替显示效果示意图

交替显示的原理很简单，首先将要显示的图形分别存放在绘图页的第 0 页及第 1 页，再将第 0 页的图形以垂直线为单位，一次搬移一条线到第 1 页即可。以下便是交替显示效果的范例程序：

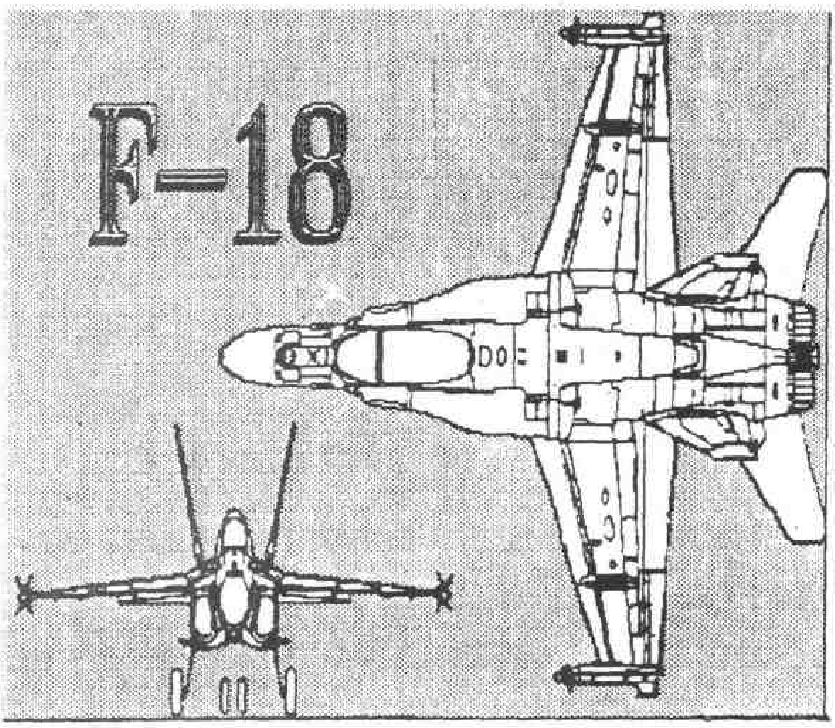
```
/* SWAP1.C */

#include "graphics.h"
#include "stdio.h"
#include "stdlib.h"
main()
{
        int driver=DETECT,mode;
        initgraph(&driver,&mode,"");
        swap();
        getch();
        closegraph();
}

swap()
{
        FILE *fp;
        unsigned i;
        char *buff;
        char *temp;
        buff=(char *)malloc(32400);
        temp=(char *)malloc(100);

        setactivepage(0);
        fp=fopen("fzr.blk","rb");
        fread( buff,1,19322,fp);
        putimage(0,0,buff,0);
        fclose(fp);

        setvisualpage(1);

        setactivepage(1);
        fp=fopen("f18.blk","rb");
        fread( buff,1,23600,fp);
        putimage(0,0,buff,0);
        fclose(fp);

        while( !kbhit() )   {
                setactivepage(1);
                getimage(0,0,719,347,buff);
```

```
            for( i=0;i<719;i++ )  {
                 setactivepage(0);
                 getimage(719-i,0,719-i,347,temp);
                 setactivepage(1);
                 putimage(718-i,0,temp,0);
            }

            setactivepage(0);
            putimage(0,0,buff,0);
            setactivepage(1);
            getimage(0,0,719,347,buff);

            for( i=0;i<719;i++ )  {
                setactivepage(0);
                getimage(i,0,i,347,temp);
                setactivepage(1);
                putimage(i,0,temp,0);
            }
            setactivepage(0);
            putimage(0,0,buff,0);
      }
      free(buff);
}
```

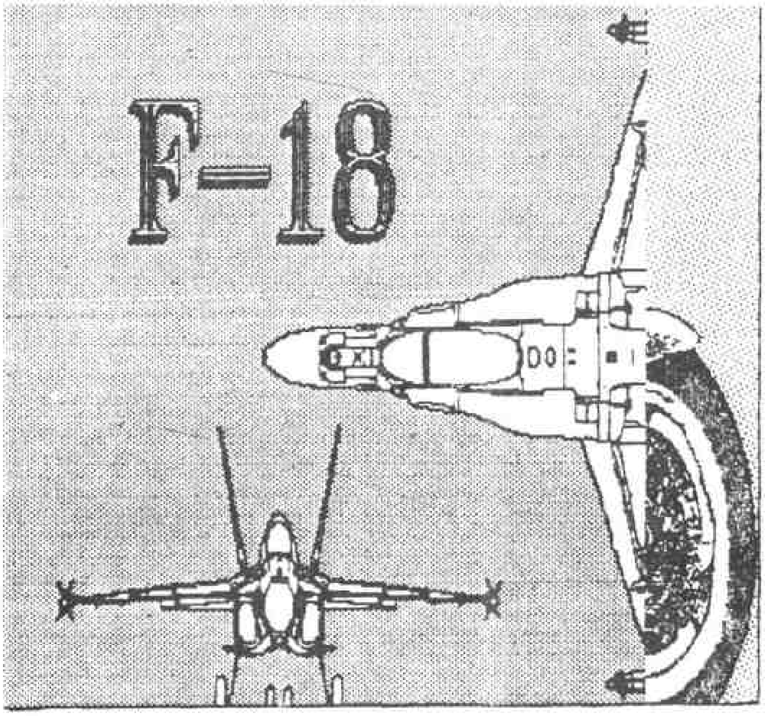


(1)

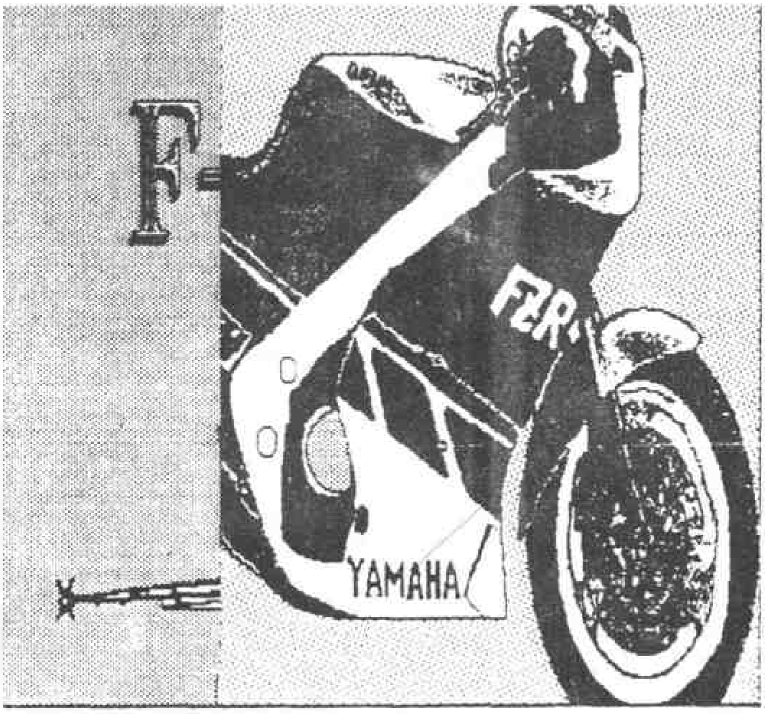
```
        for( i=0;i<719;i++ )  {
            setactivepage(0);
            getimage(719-i,0,719-i,347,temp);
            setactivepage(1);
            putimage(718-i,0,temp,0);
        }

        setactivepage(0);
        putimage(0,0,buff,0);
        setactivepage(1);
        getimage(0,0,719,347,buff);

        for( i=0;i<719;i++ )  {
            setactivepage(0);
            getimage(i,0,i,347,temp);
            setactivepage(1);
            putimage(i,0,temp,0);
        }
        setactivepage(0);
        putimage(0,0,buff,0);
    }
    free(buff);
}
```



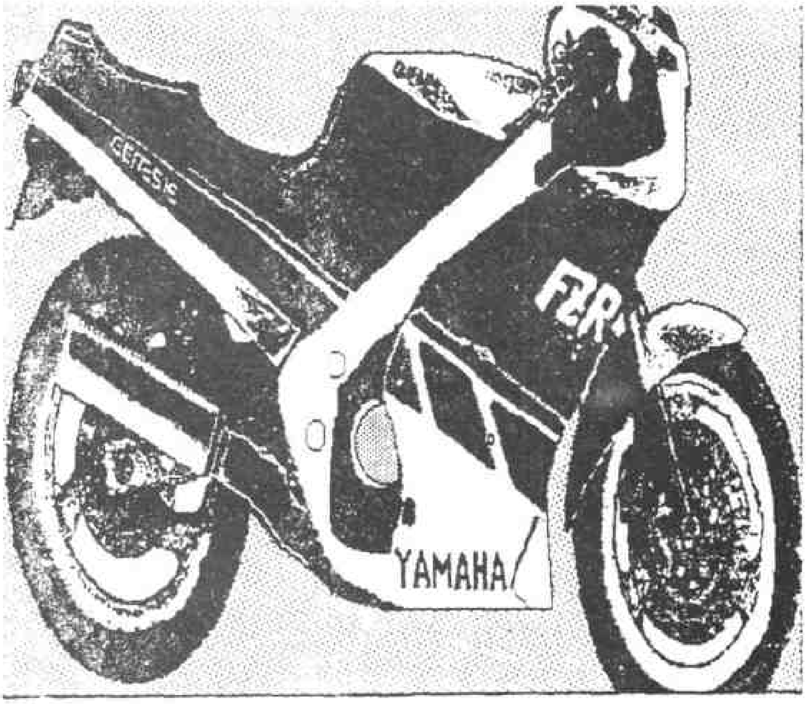
(1)



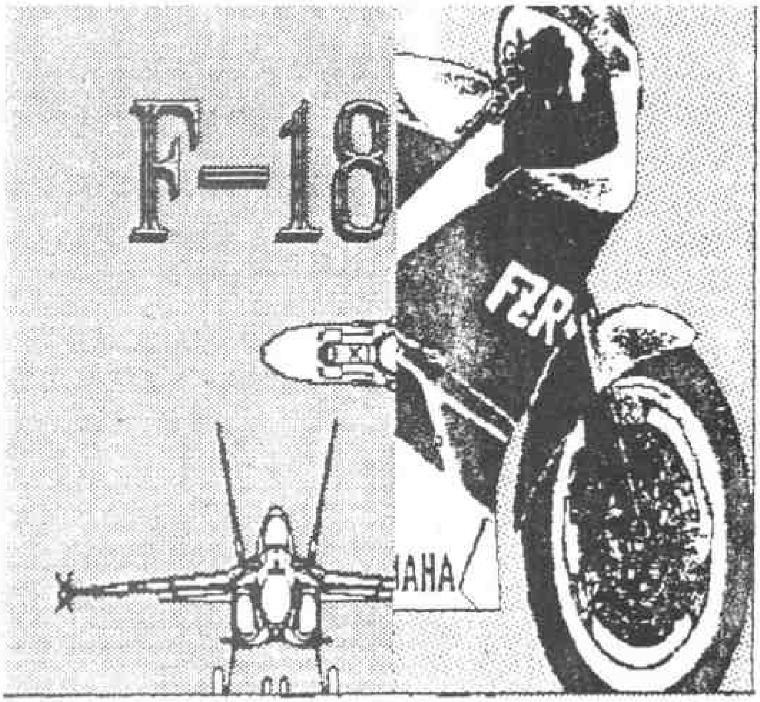
(2)



(3)



(4)



(5)

图 2-18　交替显示程序 SWAP1.C 的执行实例

2.4　实例汇聚

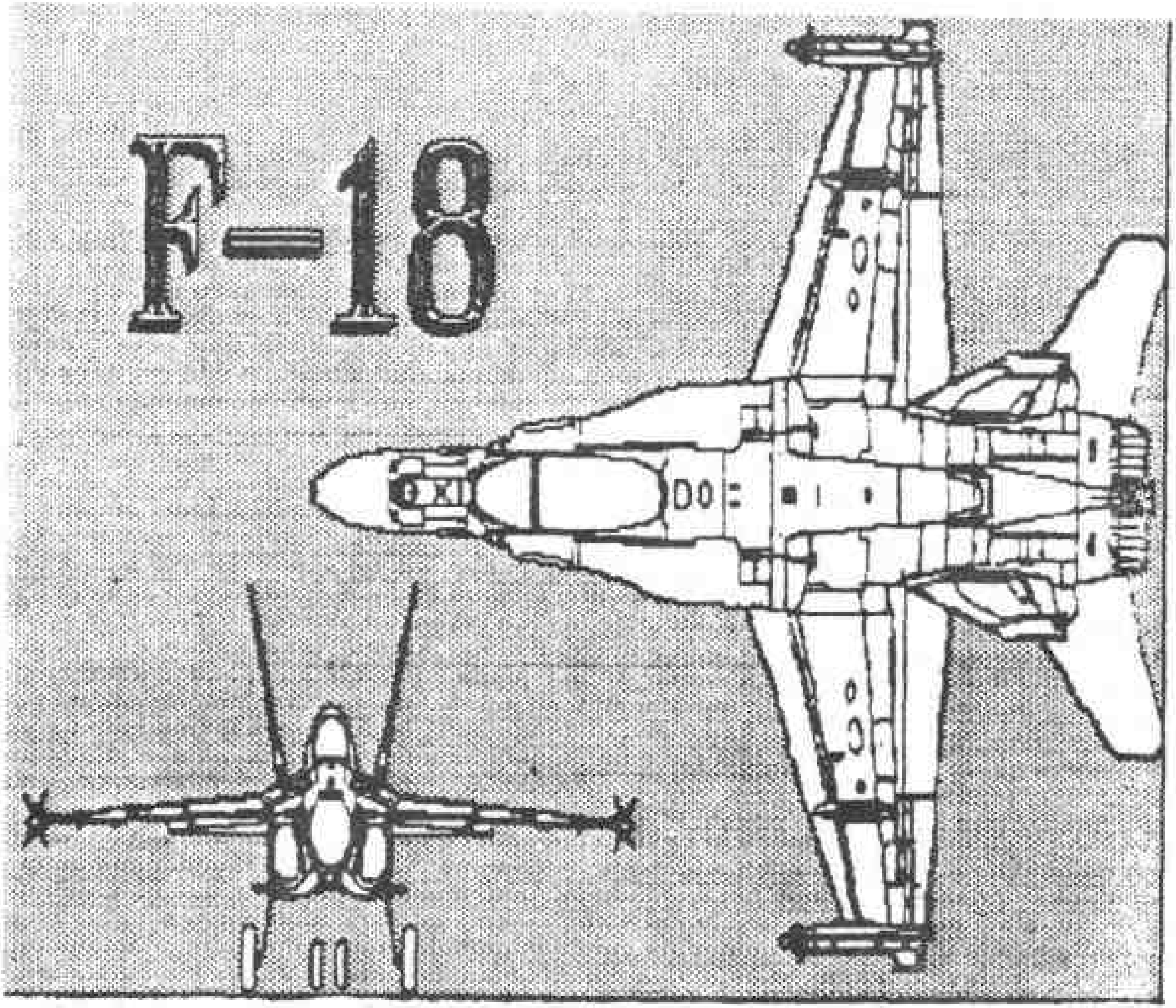
了解了上一节所介绍的各种图形显示技巧的程序设计方法后，在本节我们将以上的图形显示子程序，汇聚成为一个具有实用价值的图形显示程序。

以 C 语言的效率及编译后的执行速度，编写图形显示程序当是最恰当的选择。因此，首先我们就以 Turbo C 来引导各读者设计一个较为实用的图形显示程序。对于这个范例程序我们要求如下:

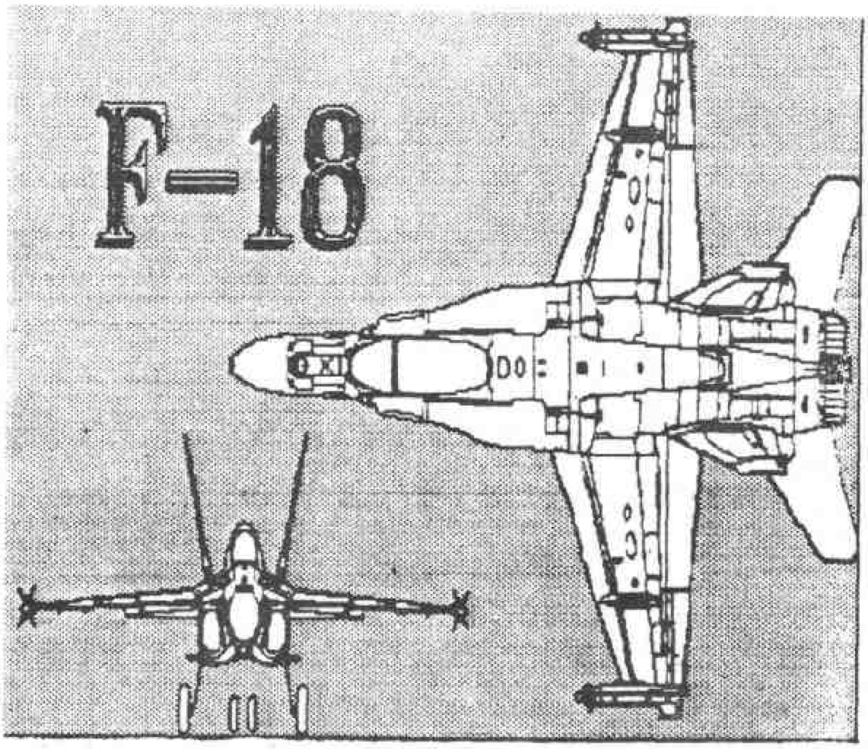
1. 必须是单、彩色两用。
2. 程序可加命令行参数两个，分别指定显示的图形及显示的效果编号(1-6)；例如:
   C>DEMO GIGER.MAP 2
3. 如果用户仅指定显示的图形，而未指定显示之编号，则六种效果全部显示。
4. 如果用户忘记指定任何参数，则出现"使用方法说明"的画面告知用户。

就以上所列的各项要求，使用 C 语言皆可轻易地完成。在单彩色两用的部分，很明显地必须准备两套显示函数，然后一进人程序即利用 Turbo C 的 detectgraph 函数判断目前的 graph__driver 为什么；若为 HERCMONO，则执行单色的显示函数；若 garph__driver 为 EGA 或 VGA，则使用彩色的显示函数，单彩色两用的问题即迎刃而解。

在命令参数方面，C 提供 argc 变量计算参数的个数（包括程序本身），*argv[ ]字符



(6)



(6)

图 2-18　交替显示程序 SWAP1.C 的执行实例

2.4　实例汇聚

了解了上一节所介绍的各种图形显示技巧的程序设计方法后，在本节我们将以上的图形显示子程序，汇聚成为一个具有实用价值的图形显示程序。

以 C 语言的效率及编译后的执行速度，编写图形显示程序当是最恰当的选择。因此，首先我们就以 Turbo C 来引导各读者设计一个较为实用的图形显示程序。对于这个范例程序我们要求如下：

1. 必须是单、彩色两用。
2. 程序可加命令行参数两个，分别指定显示的图形及显示的效果编号(1–6)；例如：
   C>DEMO GIGER.MAP 2
3. 如果用户仅指定显示的图形，而未指定显示之编号，则六种效果全部显示。
4. 如果用户忘记指定任何参数，则出现"使用方法说明"的画面告知用户。

就以上所列的各项要求，使用 C 语言皆可轻易地完成。在单彩色两用的部分，很明显地必须准备两套显示函数，然后一进入程序即利用 Turbo C 的 detectgraph 函数判断目前的 graph_driver 为什么；若为 HERCMONO，则执行单色的显示函数；若 garph_driver 为 EGA 或 VGA，则使用彩色的显示函数，单彩色两用的问题即迎刃面解。

在命令参数方面，C 提供 argc 变量计算参数的个数（包括程序本身），＊argv[ ]字符

串数组储存参数本身。例如:

DEMO TIGER.MAP2

则 argc 等于 3，而 argv[0]= "DEMO",argv[1]= "TIGER.MAP",argv[2]= "2"。利用这两个参数，本范例程序便可在进入程序后而未正式执行显示之前，先利用 argc 判定参数个数是否达到要求:

1. 如果 argc=1，表示用户并未输入参数，程序输出信息告诉用户使用方法。
2. 如果 argc=2，则令 filename 等于 argc[1]，读入此图形并显示六种效果。
3. 如果argc=2，除filename外再令demo__type=atoi(argv[2])，装入filename的图形并以 demo__type 的编号为显示依据，显示一次效果。利用 argc 及 argv，要求 2.3.4.也全部解决。

另外，显示图形之前必须把图形载入到显示页的第 1 页。而载入图形的方法本章并未介绍，所以在此先"借用"下一章的图形载入函数分别为单色的 loadampm 及彩色的 loadampc，此二函数可将扩展名为 map(Rambow)或 bpc（绘图大师）的图形载入至显示页的第 1 页以供显示，其原理将于下章介绍。

上一节所介绍的函数皆是单色函数，如要转换为彩色的函数，则需做如下的改变:

1. 分辨率由原来的 720×348 变为 640×350
2. 加入 outportb(0x3cc,5); outportb(0x3cf,1)l;
3. 原 movedata 函数改为 cmove 函数
4. 地址由 0xb800 改为 0xa800，0xb000 改为 0xa000.
5. 搬移之字节数目由 90 改为 80。

以下就是符合上述要求的范例程序 DEMO.C:

```
/ * DEMO.C * /

#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void loadmapm(char * );
void loadmapc(char * );
void showm(int);
void showc(int);
void map2(void);
void map16(void);

void pull__up2(void);
void cross__h2(void);
void cross__v2(void);
void rain2(void);
void curtain2(void);
```

```
void block2(void);

void pull_up16(void);
void cross_h16(void);
void cross_v16(void);
void rain16(void);
void curtain16(void);
void block16(void);

unsigned int addr[348];

main(argc, argv)
int argc;
char *argv[];
{
    int graph_driver, graph_mode;
    char filename[20];
    int demo_type;

    if(argc<2)      {
            printf("\a\nUsage : DEMO  <Graph_file>  <Function No.>\n\n");
            printf("Function No.:\n");
            printf("            Pull up ...... 1\n");
            printf("            Cross h ...... 2\n");
            printf("            Cross v ...... 3\n");
            printf("            Rain    ...... 4\n");
            printf("            Curtain ...... 5\n");
            printf("            Block   ...... 6\n\n");
            printf("Examples    :\n");
            printf("            DEMO  demo.map \n");
            printf("            DEMO  demo.map  4\n");
            exit(0);
    }
    else     {
        strcpy(filename, argv[1]);
        demo_type=0;
        if(argc>2) demo type = atoi(argv[2]);
    }
    detectgraph(&graph_driver, &graph_mode);

    if(graph_driver==HERCMONO)
    {
            initgraph(&graph_driver, &graph_mode, "");
            map2();
```

```
            loadmapm(filename);
            showm(demo_type);
    }
    else if((graph_driver= =EGA)||(graph_driver= =VGA))        {
          if(graph_driver= =EGA) graph_mode=EGAHI;
          else                   graph_mode=VGAMED;
          initgraph(&graph_driver, &graph_mode, "");
          map16();
          loadmapc(filename);
          showc(demo_type);
    }
    else      {
         printf("\nRequire EGA / VGA / Hercules!\n");
         exit(0);
    }
    closegraph();
}

void loadmapm(char * name)
{
    FILE  * fp;
    char  * buf;
    char low, high;
    unsigned size, x, y;
    union {
          char ch[2];
          int no;
    } value;

    if ((fp=fopen(name,"rb")) = = NULL)    {
         puts("\a");
         closegraph();
         printf("\nFile reading error!!");
         exit(1);
    }
    else        {
          if ((unsigned)(filelength(fileno(fp))) >= coreleft())
          {
              puts("\a");
              closegraph();
              printf ("\nSorry! Memory is not enough.\n");
              exit(1);
          }
          else
```

```
            size = filelength(fileno(fp));

        buf=(char  * ) malloc(size);
        setvisualpage(0);
        setactivepage(1);
        fread(buf,1,size,fp);
        value.ch[0]  =  buf[7];
        value.ch[1]  =  buf[8];
        x  =  value.no;
        value.ch[0]  =  buf[9];
        value.ch[1]  =  buf[10];
        y  =  value.no;
        x  =  (720-x) / 2;
        y  =  (348-y) / 2;
        clearviewport();
        buf[7]--; buf[9]--;
        putimage(x,y,buf+7,COPY_PUT);
        fclose(fp);
        free(buf);
    }
}

void loadmapc(char  * name)
{
    FILE  * fp;
    int  i,  j;
    char  picdata;
    unsigned  size,  x,  y,  low,  high;

    if((fp  =  fopen(name,  "rb")) = =NULL) {
        puts("\a");
        closegraph();
        printf("\nFile reading error!!");
        exit(1);
    }
    for(i = 1;  i < = 7;i++)
        fgetc(fp);
    low  =  fgetc(fp);
high  =  fgetc(fp);
    x  =  (unsigned  int)  low  +  (unsigned  int)  high  *  256;
    low  =  fgetc(fp);  high  =  fgetc(fp);
    y  =  (unsigned  int)  low  +  (unsigned  int)  high  *  256;
    if((x%8)= =0)
    x=x / 8
```

```
        else
            x = ((x & 0xfff8) + 8)/8;
        for(i=0;i<y;i++)  {
            outportb(0x3c4, 2); outportb(0x3c5, 1);
            for(j=0;j<x;j++)
                pokeb(0xa800,i*80+j,fgetc(fp));

            outportb(0x3c4, 2); outportb(0x3c5, 2);
            for(j=0;j<x;j++)
                pokeb(0xa800,i*80+j,fgetc(fp));

            outportb(0x3c4, 2); outportb(0x3c5, 4);
            for(j=0;j<x;j++)
                pokeb(0xa800,i*80+j,fgetc(fp));

            outportb(0x3c4, 2); outportb(0x3c5, 8);
            for(j=0;j<x;j++)
                pokeb(0xa800,i*80+j,fgetc(fp));
        }
        outportb(0x3c4, 2); outportb(0x3c5, 0xf);
        fclose(fp);
}

void map2()
{
    register int i, j;

    for(i=0; i<348; i++)
        addr[i] = 8192*(i%4) + 90*(i/4);
}

void map16()
{
    register int i;

    for(i=0; i<350; i++)
        addr[i] = i*80;
}

void showm(int type)
{
    int i, start, end;

    setactivepage(0);
```

```
    if(!type)   start=1, end=6;
    else        start=end=type;
    for(i=start; i<=end; i++)       {
        switch(i)   {
            case 1 : pull_up2();
                     break;
            case 2 : cross_h2();
                     break;
            case 3 : cross_v2();
                     break;
            case 4 : rain2();
                     break;
            case 5 : curtain2();
                     break;
            case 6 : block2();
        }
        getch();
        clearviewport();
    }
}

void showc(int type)
{
    int i, start, end;

    setactivepage(0);
    if(!type)   start=1, end=6;
    else        start=end=type;
    for(i=start; i<=end; i++)       {
        switch(i)       {
            case 1 : pull_up16();
                     break;
            case 2 : cross_h16();
                     break;
            case 3 : cross_v16();
                     break;
            case 4 : rain16();
                     break;
            case 5 : curtain16();
                     break;
            case 6 : block16();
        }
        getch();
        clearviewport();
```

```
    }
}

void pull_up2()
{
    int i, j;

    for(i=347; i>=0; i--)
    for(j=0; j<348-i; j++)
        movedata(0xb800, addr[j], 0xb000, addr[i+j], 90);
}

void cross_h2()
{
    int i, j;

    for(i=1; i<=90; i++)
    for(j=0; j<348; j+=2)    {
        movedata(0xb800, addr[j]+90-i, 0xb000, addr[j], i);
        movedata(0xb800, addr[j+1], 0xb000, addr[j+1]+90-i, i);
    }
}

void cross_v2()
{
    int i,j;

    for(i=0;i<347;i+=2)
    for(j=i;j>=0;j-=2)    {
        movedata(0xb800,addr[346-i+j],0xb000,addr[j],90);
        movedata(0xb800,addr[i-j+1],0xb000,addr[347-j],90);
    }
}

void rain2()
{
    int i, j;

    for(i=347;i>=0;i--)
    for(j=0;j<=i;j++)
        movedata(0xb800, addr[i], 0xb000, addr[j], 90);
}

void curtain2()
```

```
{
    int i, j;

    for(i=0;i<=28;i++)    {
        delay(30);
    for(j=0;j<=11;j++)
        movedata(0xb800,addr[29 * j+i],0xb000,addr[29 * j+i],90);
    }
}

void block2()
{
    int i, j, k;

    for(i=344; i>=0; i-=4)
    for(j=1; j<=i; j++)    {
    for(k=0; k<4; k++)
        movedata(0xb800,addr[i+k],0xb000,addr[j+k],90);
    for(k=0; k<90; k++)
        pokeb(0xb000, addr[j-1]+k, 0);
    }
}

void pull_up16()
{
    int i, j;

    outportb(0x3ce, 5); outportb(0x3cf, 1);
    for(i=349; i>=0; i--)
    for(j=0; j<350-i; j++)
        cmove(0xa800, addr[j], 0xa000, addr[i+j], 80);
}

void cross_h16()
{
    int i, j;

    outportb(0x3ce, 5); outportb(0x3cf, 1);
    for(i=1; i<=80; i++)
    for(j=0; j<350; j+=2) {
        cmove(0xa800, addr[j]+80-i, 0xa000, addr[j], i);
        cmove(0xa800, addr[j+1], 0xa000, addr[j+1]+80-i, i);
    }
}
```

```
void cross__v16()
{
     int i,j;

     outportb(0x3ce,5);outportb(0x3cf,1);
    for(i=0;i<348;i+=2)
     for(j=i;j>=0;j-=2)          {
           cmove(0xa800,addr[346-i+j],0xa000,addr[j],80);
           cmove(0xa800,addr[i-j+1],0xa000,addr[347-j],80);
    }
}

void rain16()
{
    int i, j;

    outportb(0x3ce, 5);
    outportb(0x3cf, 1);
    for(i=349;i>=0;i--)
    for(j=0;j<=i;j++)
          cmove(0xa800, addr[i], 0xa000, addr[j], 80);
}

void curtain16()
{
    int i, j;

    outportb(0x3ce,5);outportb(0x3cf,1);
    for(i=0;i<=28;i++)           {
          delay(30);
          for(j=0;j<=11;j++)
          cmove(0xa800,addr[29 * j+i],0xa000,addr[29 * j+i],80);
    }
}

void block16()
}
    int i, j, k;

    outportb(0x3ce,5);outportb(0x3cf,1);
    for(i=340; i>=0; i-=10)
    for(j=1; j<=i; j++)        {
    for(k=0; k<10; k++)
```

```
372:            cmove(0xa800,addr[i+k],0xa000,addr[j+k],80);
373:        for(k=0; k<80; k++)
374             pokeb(0xa000, addr[j-l]+k, 0);
375:        }
376:}
```

程序说明

39—52　列出使用说明.

58　判断屏幕为单色或彩色.

82—128　载入单色图形.

130—171　载入彩色图形.

191—215　显示 16 色效果.

217—241　显示 16 色效果.

243—250　显示单色上拉效果.

252—261　显示单色水平推拉效果.

263—272　显示单色垂直推拉效果.

274—281　显示单色雨状效果.

283—292　显示单色百叶窗效果.

294—305　显示单色积木效果.

307—315　显示彩色上拉效果.

317—327　显示彩色水平推拉效果.

329—339　显示彩色垂直推拉效果.

341—350　显示彩色雨状效果.

352—362　显示彩色百叶窗效果.

364—376　显示彩色积木效果.

2.5 结　论

图形显示的应用非常广泛，在一般性的大型软件中，大多少不了它．其实图形显示的方法很多，除了本章所介绍的推拉、上下交错、左右交错、百叶窗、雨状、积木之外，常见的方法还有淡出淡入、雾状、马赛克、幅射呈现、镭射光模拟等等多种显示效果．有兴趣的读者，不妨多写程序，相信你一定会能够开发出更新、更动人的图形显示方法．

第三章　图形文件格式

3.1　简　介

几乎所有的绘图软件都提供了储存图形的功能，以便用户本次所画的图形得以在下次取出编辑或显示。由于每一套绘图软件都拥有属于自己的图形储存格式，所以往往在甲软件产生的图形，却无法直接在乙软件上读出。而偏偏又没有一套绘图软件能够完全符合用户需求，也许用户习惯使用甲软件来扫描图形，使用乙软件来写中文字，使用丙软件来画图……等等。此时，了解图形文件格式的结构，从而使用现有的文件格式转换工具，甚至自行设计，便成为相当重要的课题。

本章将详论市面上许多著名绘图软件的图形文件格式结构、读文件及解码方法；每种格式都附有 C 语言的读文件、显示范例以供应用。不论是单纯地显示图形，或者是编写格式转换工具，都可以在此获得必备的相关知识。

3.1.1　位映射式与向量式图形文件

图形文件若以储存方式区分，有向量式及位映射式两大类。向量式图形文件中所储存的数据，是一连串的图形描述指令或语句，利用这些指令或语句来描述该系统所建立的图形。通常想重现此图形，必须有一个绘图的核心程序来负责解释这些指令，并将它画在屏幕上。由于这种绘图系统较为复杂，又须费心去构造一个绘图核心程序，因此本章不作介绍，而将所有的注意力集中在另一种储存方式——位映射式图形文件上。

所谓位映射式，顾名思义，就是将屏幕上（或绘图软件的"画布"上）的图形数据原原本本地"映射"到文件上，使得文件里的数据其实就是屏幕上图形数据的一份复制。这样的储存方式，使得文件的存取变得更为直接，也较为简便，然而却比较浪费磁盘空间。因为一般的单色屏幕的分辨率为 720×348，那么储存一张全屏幕的图形需要：

$(720/8)\times 348=31320$ 字节

约 30.5KB 的磁盘空间。如果是彩色的图形，以常见的 640×480 分辨率，16 种色彩的模式来计算：

$(640/8)\times 480\times 4=153600$ 字节

共占 150KB 的空间。由此可见，虽然位映射式图形文件较为直接，但是基于储存介质空间上的考虑，除了小张的区块图形外，对于较大张的图形，通常在储存及读取之前，必先经过一道压缩／展开的手续，如图 3-1 所示。

经过了压缩的过程后，图形在储存介质上所占的空间就可以小得多了。然而也是因为这一道手续，使得各图形文件较难统一应用。因为各种图形文件压缩的方式不尽相同，除非熟知其方法，否则根本无法读取。本章的下面几节将详细地解释这些算法。在讨论这些文件结构及算法之前，让我们先来解释位映射式图形是如何储存在文件里的。

图 3−1　大图形的显示

3.1.2　图形储存方式

单色图形的每一个像素(pixel)只有亮及不亮两种变化，因此每个图形点可以轻易地以一个 bit 来表示，由于个人计算机通常都以 byte 为基本的处理单位，所以当我们存取这些图形数据的一个 byte 时，实际上是处理了 8 个像素。假设有一张如下的图形：

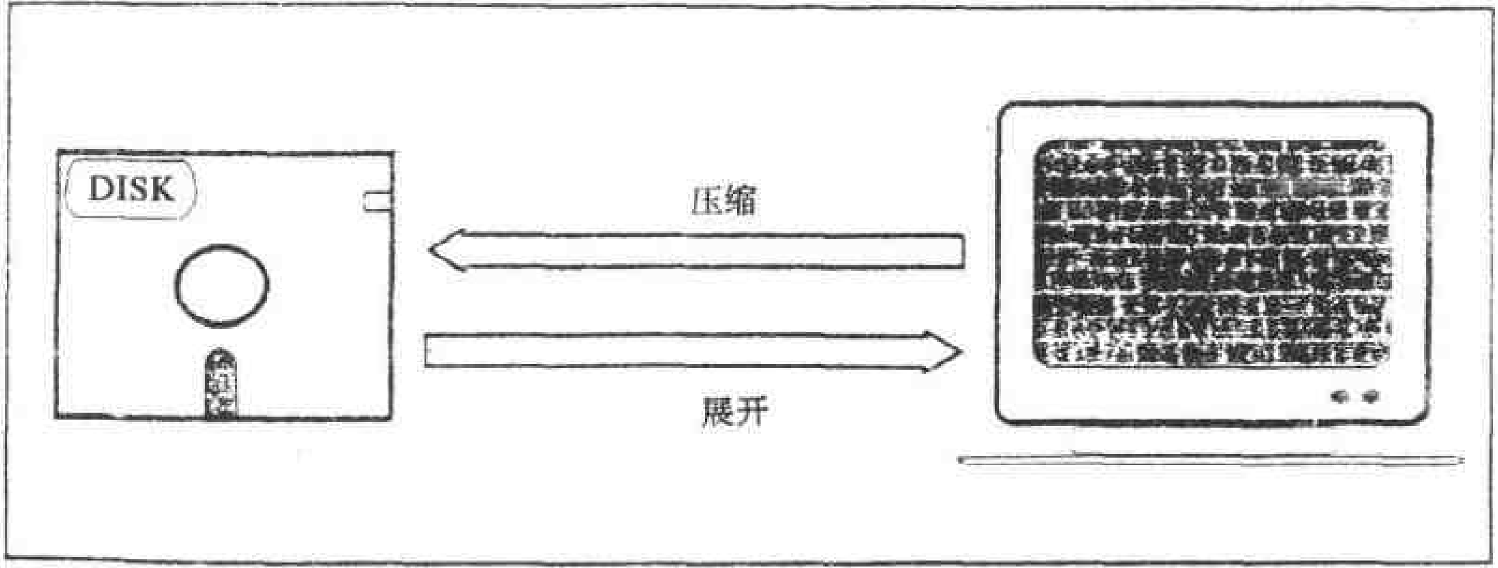
图形	对应的 byte 值
○ ○ ○ ● ● ○ ○ ○	18h
○ ○ ● ● ● ● ○ ○	3Ch
○ ● ● ● ● ● ● ○	7Eh
○ ○ ○ ● ● ○ ○ ○	18h
○ ○ ○ ● ● ○ ○ ○	18h
○ ○ ○ ● ● ○ ○ ○	18h
○ ○ ○ ● ● ○ ○ ○	18h
○ ○ ● ● ● ● ○ ○	3Ch

则在储存时，将会以下列的字串储存于磁盘上：

·　18h　3ch　7Eh　18h　18h　18h　18h　3ch

图形数据的排列是以行（扫描线）为主，第一行的数据由左至右储存完毕后，接着储存第二行的数据，直到全部储存完为止。假如想要储存全页的单色屏幕，也是先储存第一条扫描线的 90 字节，接着再储存第二条扫描线的 90 字节，依此类推。然而由于 Hercules 的硬件本身的限制（详见第一章），屏幕上扫描线的图形数据储存于显示内存之位置并非连续排列，所以在存取数据之前，必须先执行一个制作位置对照表的函数，产生一个扫描线起始内存地址的对照表格。如此一来，我们就可以利用这表格直接处理扫描线的号码，而不用去担心实际上该扫描线存在内存的什么地址了。

下面这段程序码示范了如何利用查表法来将 90 个 byte 的数据写入扫描线第 5 行：


DISK
压缩
展开

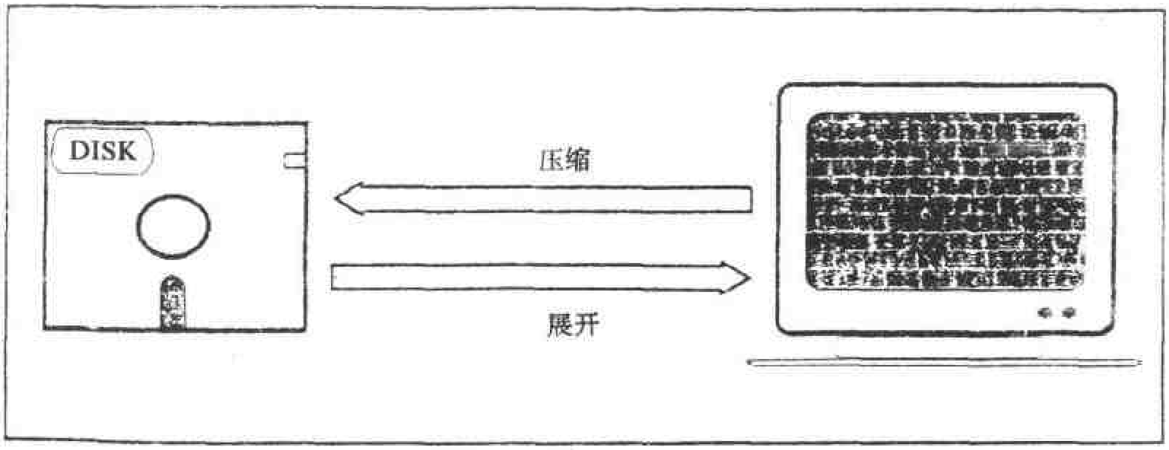


图 3-1 大图形的显示

3.1.2 图形储存方式

单色图形的每一个像素(pixel)只有亮及不亮两种变化，因此每个图形点可以轻易地以一个 bit 来表示，由于个人计算机通常都以 byte 为基本的处理单位，所以当我们存取这些图形数据的一个 byte 时，实际上是处理了 8 个像素。假设有一张如下的图形：

图形	对应的 byte 值
○ ○ ○ ● ● ○ ○ ○	18h
○ ○ ● ● ● ● ○ ○	3Ch
○ ● ● ● ● ● ● ○	7Eh
○ ○ ○ ● ● ○ ○ ○	18h
○ ○ ○ ● ● ○ ○ ○	18h
○ ○ ○ ● ● ○ ○ ○	18h
○ ○ ○ ● ● ○ ○ ○	18h
○ ○ ● ● ● ● ○ ○	3Ch

则在储存时，将会以下列的字串储存于磁盘上：

18h 3ch 7Eh 18h 18h 18h 18h 3ch

图形数据的排列是以行（扫描线）为主，第一行的数据由左至右储存完毕后，接着储存第二行的数据，直到全部储存完为止。假如想要储存全页的单色屏幕，也是先储存第一条扫描线的 90 字节，接着再储存第二条扫描线的 90 字节，依此类推。然而由于 Hercules 的硬件本身的限制（详见第一章），屏幕上扫描线的图形数据储存于显示内存之位置并非连续排列，所以在存取数据之前，必须先执行一个制作位置对照表的函数，产生一个扫描线起始内存地址的对照表格。如此一来，我们就可以利用这表格直接处理扫描线的号码，而不用去担心实际上该扫描线存在内存的什么地址了。

下面这段程序码示范了如何利用查表法来将 90 个 byte 的数据写入扫描线第 5 行：

```
unsignedd  addr[348];
main()
{
    map2();
    for (i=0;i<90;i++)
    pokeb (0xb000, addr[5]+i,0xff);
}
```

而单色 Hercules 的制表程序如下:

```
unsigned addr[348];
maps2()
{
    regisgter int i;
    for (i=0;i<348;ii++)
      addr[i]=8192 * (i%4)+90 * (i / 4);
}
```

3.2 各种图形文件的格式

3.2.1 Turbo C 的图形储存格式

在介绍各种文件格式之前，让我们先来了解 Turbo C 究竟是以何种格式来储存 getimage 函数所取得的图形数据．首先我们写一段程序来观察其内存内容:

```
/ * TCIMASGE.C * /

#include <graphics.h>
#include <stdio.h>
#include <alloc.h>

main()
{
        int graph__driver=DETECT, graph__mode;
        char * buf;
        unsigned size;
        int i;

        initgraph(&graph__driver,&graph__mode,"");
        setfillstyle(SOLID__FILL,1);
        bar(0,0,15,15);
        size = imagesize(0,0,15,15);
        buf = (char * ) malloc(size);
        getimage(0,0,15,15,buf);
```

```
    putimage(100,100,buf,COPY_PUT);
    getch();
    closegraph();
    for(i=0;i<size;i++)
        printf("%4x,buf[i]&0xff);
    printf("\nSize = %d\n",size);
    getch();
}
```

这段程序主要分成三部分:

1. 在(0,0)处画一个 16×16 大小的矩形。
2. 将该矩形getimage后，于(100,100)再putimage出来，查看所取得的图形是否完整。
3. 按任意键后，在屏幕上方打印出内存内容及imagesize函数所计算得到的值。如下:

```
 f  0  f  0  ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff  0  0
size = 38
```

imagesize 计算所得的大小为 38，而真正储存 16×16 大小矩形的图形数据仅须 (16/2)*16=32 字节，在多出来的 6 个 bytes 中，其中 4 个 bytes 表示图形的宽度减 1 及高度减 1，而剩下的 2 个 bytes 则为 Turbo C 的保留字节。我们将它的格式整理如表 3-1:

表 3-1 putimage 图形的格式

起始字节	长度（byte)	内容
0	2	图形的宽度减 1
2	2	图形的高度减 1
4	图形实际长度	图形数据
倒数第二个	2	保留

了解 Turbo C 的 getimage 的图形数组储存方式后，接着就可以将各种图形文件转成 putimage 可接受的格式，然后用 putimage 将它显示于屏幕。以下各节即是各种图形文件的格式及读取方法。

3.2.2 BASIC 中 BSAVE 指令的储存格式

许多人第一次设计程序所使用的语言便是 BASIC。近两年来，Quick Basic 渐渐成为热门的语言，本书虽然以 C 语言为讨论重点，但考虑到有时也会应用 Basic 的图形文件，因此在本章中也对 Basic 语言的图形文件格式稍做说明。

首先我们在 BASIC 下写一段程序，画一个简单的 16×16 大小的矩形，然后将此矩形用 BSAVE 指令储存于磁盘上，为了确定是否正确储存，再用 BLOAD 指令将矩形读出，放在屏幕上比较和原图是否一致．程序 TEST.BAS 的列表如下:

```
10 OPTION BASE 1
20 SCREEN 2: CLS :KEY OFF
30 DIM G1%(30),G2%(30)
40 LINE(100,100)-(115,115),,B
50 GET(100,100)-(115,115),G1%
60 PUT(200,200),G1%,PSET
70 BSAVE "test.map",VARPTR(G1%(1) ),36
80 BLOAD "test.map",VARPTR(G2%(1) )
90 PUT(200,200),G2%,PSET
100 END
```

执行这个程序将会在屏幕上看到三个一模一样的矩形，同时会将这个矩形以 test.map 为文件名储存在磁盘上．接下来我们就以 DEBUG 程序观察 test.map 的内容:

```
C>debug test.map
-r
AX=0000 BX=0000 CX=002C DX=0000 SP=01BE BP=0000 SI=0000 DI=0000
DS=1696 ES=1696 SS=1696 CS=1696 IP=0100 NV UP EI PL NZ NA PO NC
1696:0100 FD            STD
-d
1696:0100  FD BE 21 04 0C 24 00 10-00 10 00 FF FF 80 01 80   ..!.. $..........
1696:0110  01 80 01 80 01 80 01 80-01 80 01 80 01 80 01 80   ................
1696:0120  01 80 01 80 01 80 01 80-01 FF FF 1A F5 ED2C 53    ................S
1696:0130  8B 1E E8 05 5E 87 F3 56-E8 B9 44 D6 51 E5 ED53    ........V......)S
1696:0140  E8 2E FE87 DA59 5B 58-04 E8 FE C8 51 B9 4C 1D     .>...Y[XQ.D.Q.L.
1696:0150  51 50 52 E8 32 FE 5A 58-8A E8 FE C8 8A C8 3A 07   QPR.2.ZX......:.
1696:0160  B0 00 72 01 C3 8B F2 8A-04 0A C0 8A C5 75 01 C3   ..r..........u..
1696:0170  8A 07 43 8A 2F 43 8A 3F-8ADD B5 00 03 D9 2A C1    ..C./ C.?...... *.
```

在执行 DEBUG 并载人 test.map 之后，首先使用 r 指令观看其寄存器，其中 CX 的内容就是文件的长度．请注意！在 DEBUG 中的所有的数字都是十六进制表示，所以 CX 的内容 2c 换算成十进制为 44，也就是文件全长为 44 个 bytes.

在此让我们回想一下，储存一个 16×16 大小的图需要多少 bytes．因为在单色模式下每 bytes 可以表示 8 个像素，所以 16×16 大小的图形需要(16／8) * 16＝32 个 bytes。然而除了图形数据本身外，尚需要 2 个整数表示宽和高，每个整数各需占 2 个 bytes，所以表示 16×16 的图形共需:

4+(16／8) * 16＝36 字节

回头看一下 BASIC 程序的行号 70，储存长度就是 36.

文件全长 44 字节，而实际上表示图形数据只需 36 字节，那么多出来的 8 个 byte 的作用是什么？请参考上图，其中被框住的部分是文件的内容，而用底线标出的，就是多出

来的 8 个 byte，分别为：

FD　BE　21　04　0C

24　00

1A

其中：

FD	为识别记号，在此固定为FD。
BE 21	为在BASIC的环境中，BSAVE的当前的数据段地址。
04 0C	为在BASIC的环境中，BSAVE的当前的数据偏移地址。
24 00	这个值十进位为00 * 256+36＝36，代表实际表示图形数据的长度，包括长、宽及图形数据本身。
1A	为文件结束识别记号。

根据以上分析，我们再将 BASIC 语言 BSAVE 后的文件格式列表如下：

表 3–2　BASIC 语言的 BSAVE 图形文件格式

起始字节	长度(Byte)	描　　述
0	1	文件识别记号，恒为 FD
1	2	段位址
3	2	偏移位址
5	2	图形数据全长
7	2	图形的宽度（以象素为单位）
9	2	图形的高度（以扫描线为单位）
11	全长–4	图形数据
最后一个	1	结束识别记号，恒为 1A

请将表 3–2 拿来和 Turbo C 语言的 getimage 函数所产生的位图形数组储存格式比较，其实 Turbo C 的格式只是 BASIC 语言格式的子集。如果想使用 C 语言将 BASIC 的 BSAVE 文件的图形读出，只需依照下列步骤即可完成：

1. 使用 fopen 打开文件。
2. 配置以 buf 为起始地址的内存。
3. 利用 fgetc(fp)丢弃前 7 字节。
4. 使用 fread 将图形文件读到 buf 所指定的地址。
5. buf[0]减 1
6. buf[1]减 1
7. putimage(x,y,buf,COPY_PUT)

以上七个步骤可以很轻易地利用 C 语言完成。下面，我们先以一 BASIC 的程序画一张简单的图形，并利用 BSAVE 存储文件，然后再用 C 语言将其读出。程序 DRAW.BAS 如下：

```
10 KEY OFF : CLS : SCREEN 2
```

```
20 'Draw a picture
30 FOR X = 0 TO 360 SETP 5
40  CIRCLE(X+200,SIN(X * 3.14159 / 180) * 100+160),50
50 NEXT X
60 'Save the picture
70 DIM G%(15700)
80 GET(0,0)-(719,347),G%
90 DEF SEG
100 BSAVE"SINWAVE.BMP",VARPTR(G%(0) ), 90 * 348+4
110 END
```

本程序会产生以圆圈构成的正弦波图形，并以 SINWAVE.BMP 为文件名储存在磁盘上.

```
/ *  LOAD__BAS.C * /

#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>

void load__bas(int, int, char *, int, int);

main()
{
    int graph__driver=HERCMONO, graph__mode=HERCMONOHI;

    initgraph(&graph__driver, &graph__mode, "");
    load__bas(0,0,"demo.bmp",0,COPY__PUT);
    getch();
    closegraph();
}

void load__bas(int x,int y,char *filename,int page,int status)
{
    int *buf;
    unsigned size;
    FILE *fp;
    int i;

    if((fp = fopen(filename, "rb"))= =NULL) {
       closegraph();
       puts("File reading error!");
```

```
        exit(1);
    }
    for(i=0; i<7; i++)
        fgetc(fp);
    size = filelength(fileno(fp));
    buf = (int *) malloc(size);
    if(buf==NULL) {
            closegraph();
            puts("Memory is not enough!");
            exit(1);
    }
    fread(buf, 1, size, fp);
    fclose(fp);
    buf[0]--;
    buf[1]--;
    setactivepage(page);
    putimage(x,y,buf,status);
    free(buf);
}
```

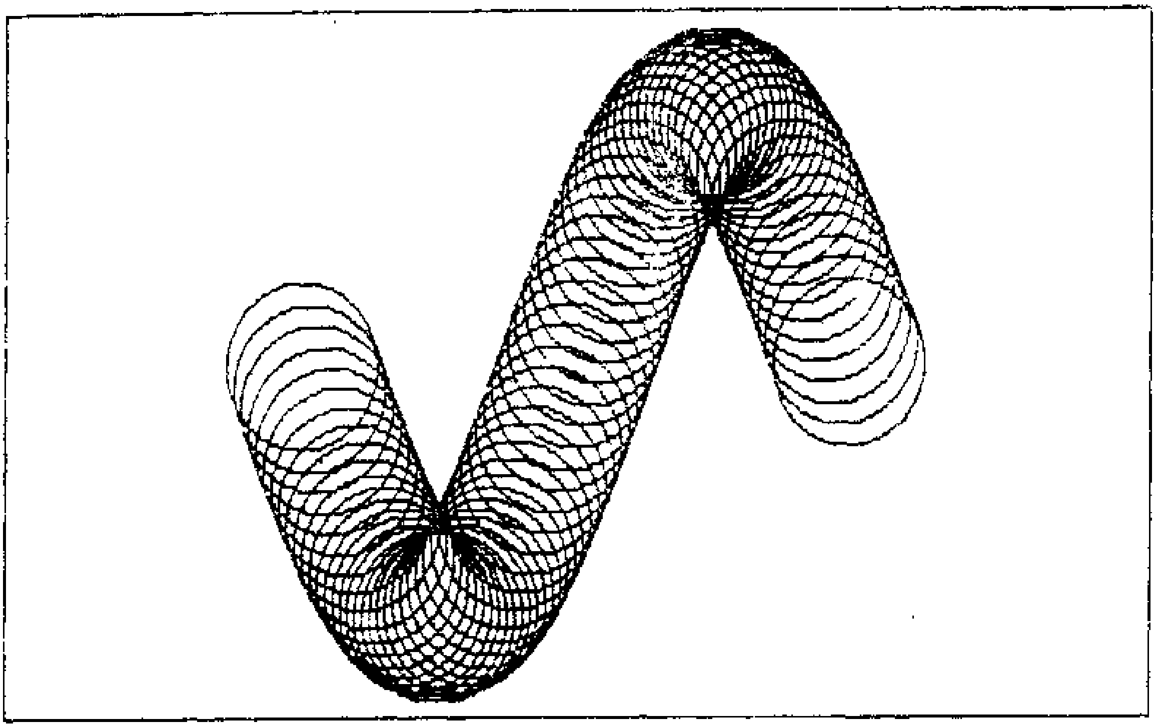


图 3-2 程序 LOAD_BAS.C 的执行结果

3.2.3 PUT 文件格式读取（变影 72）

在此必须先说明一点，就是世界上使用的图形文件格式可能不下数百种，，它们仅仅用扩展名短短的三个字母来区别，难免会有不同格式却使用相同扩展名的情形出现。例如巧匠的图形文件就和 C／GEM 软件上的图形文件名同为 TMG，但格式不同，因此本章

将在各种格式的标题之后加上括号，注明本文件是由那一套软件所产生的，以免产生混淆。

PUT 是变影 72 的区块图形文件，所谓的区块图形，在变影 72 手册上称之为小图，即是有别于全页工作图（手册上称之为大图）的称呼，是指储存工作图的部分区块而言。由于区块图形通常都很小，大抵上不超过一个屏幕，所以大部分的区块图形文件都不会经过压缩的手续，因此结构就特别简单，相当容易读取。以下就是一个 15×16 大小的实心矩形，实际上利用 PUT 格式储存后的内容，以 DEBUG 观察的结果：

```
-r
AX=0000 BX=0000 CX=0024 DX=0000 SP=01BE  BP=0000 SI=0000 DI=0000
DS=1696 ES=1696 SS=1696 CS=1696 IP=0100  NV UP EI PL NZ NA PO NC
1696:0100 FD            STD
-d
1696:0100  0F 00 01 00 FF FE FF FE-FF FE FF FE FF FE FF FE   ................
1696:0110  FF FE FF FE FF FE FF FE-FF FE FF FE FF FE FF FF   ................
1696:0120  FF FE FF FE 07 A1 6B 59-2D 48 00 8B 16 7B 37 D1   .....KY-H...{7.
1696:0130  00 7F 00 00 7F FF FF FF-04 7D 05 8B 04 A3 E8 1B   .........}.....
1696:0140  A1 EC 1B 39 05 7E 05 8B 05 A3 EC 1B 8B 5E E0 A1   ...9.........
1696:0150  EA 1B 39 07 7D 05 8B 07-A3 EA 1B 8B 5E E2 39 17   ..9.}........9.
1696:0160  7E 02 8B 17 83 C6 02 83-C7 02 83 46 E0 02 83 46   ..........F...F
1696:0170  E2 02 E2 C0 89 16 EE 1B-A1 EC 1B 2B 06 E8 1B 99   ...........+....
```

请注意 CX 的内容即为文件的实际长度，因为是十六进制，所以长度应为 36，而用线框住的部分即为文件的内容。本格式相当简单明了，前 2 个 bytes 为图形的宽，是为 15；接着的 2 个 bytes 为图形的高，是为 16；接下来的全部都是图形数据。PUT 格式可以用下面的表格来表示：

表 3-3　PUT 图形文件的格式

起始字节	长度(byte)	描述
0	2	图形的宽度
2	2	图形的高度
4	全长-4	图形数据

有一点读者可能容易发生混淆的地方在此提出来说明，即是在表示宽度和高度的地方，其值分别为：

0F　　00　　及　　10　　00

而其真正的数值化成十进位应为：

0F+00＊256=15　及　10+00＊256=16

它们的第一个 byets 是低字节，而第二个 bytes 为高字节，这是 PC 的数据储存习

惯，读者应特别加以注意．也就是说如果有一个数值为 05FF，储存于文件上则变成 FF 05 顺序的 2 个字节，应当注意。

拿 PUT 的格式表和 C 语言的比较，发现基本格式一样，只是有几个细节必须加以修改，才能由 putimage 使用：

1. 将宽度的值减 1。
2. 将高度的值减 1。
3. 无须理会保留的 2 个字节，只须保留足够的图形储存空间即可。

根据上列三个步骤写成的文件读取程序，基本上和上一小节的 LOAD_BAS.C 差别不大，仅在于本程序并没有跳过前 7 个 bytes 的动作，并且由于此处指针 * buf 的类型说明为整数，因此系统会帮我们将图宽及图高转换好，我们不必理会字节的顺序高低的问题。

下面就是程序 LOAD_PUT.C 的源程序列表。

```
/ *  LOAD_PUT.C  * /

#include <graphics.h>
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <alloc.h>

void load_put(int, int, char *, int, int);

main()
{
     int graph_driver=HERCMONO, graph_mode=HERCMONOHI
     initgraph( &graph_driver, &graph_mode, "" );
     load_put( 0,0,"demo.put",0,COPY_PUT );
     getch();
     closegraph();
}

void load_put(int x,int y,char * filename,int page,int status)
{
     int * buf;
     unsigned size;
     FILE * fp;
     int i;

     if((fp = fopen(filename, "rb"))==NULL) {
         closegraph();
         puts("File reading error!");
```

```c
        exit(1);
    }
    size = filelength(fileno(fp));
    buf = (int *) malloc(size);
    if(buf==NULL) {
        closegraph();
        puts("Memory is not enough!");
        exit(1);
    }
        fread(buf, 1, size, fp);
        fclose(fp);
        buf[0]--;
        buf[1]--;
        setactivepage(page);
        putimage(x,y,buf,status);
        free(buf);
}
```

图 3-3 程序 LOAD_PUT.C 程序的执行结果

3.2.4 MAP 文件格式的读取（Rambow PAINT，神偷）

MAP 是绘图软件 Rambow PAINT 所使用的区块文件格式，此种格式的优点是可以

```
        exit(1);
    }
    size = filelength(fileno(fp));
    buf = (int * ) malloc(size);
    if(buf= =NULL) {
        closegraph();
        puts("Memory is not enough!");
        exit(1);
    }
        fread(buf, 1, size, fp);
        fclose(fp);
        buf[0]--;
        buf[1]--;
        setactivepage(page);
        putimage(x,y,buf,status);
        free(buf);
}
```

图 3–3 程序 LOAD_PUT.C 程序的执行结果

3.2.4 MAP 文件格式的读取（Rambow PAINT，神偷）

MAP 是绘图软件 Rambow PAINT 所使用的区块文件格式，此种格式的优点是可以

直接由 BASIC 的 BLOAD 载入显示，而无须经过任何转换的手续。由此可知，本格式一定和 BASIC 的 BSAVE 文件相似，但其详细内容怎样呢？我们一样使用 DEBUG 程序来观察一个小 MAP 文件：

```
-r
AX=0000 BX=0000 CX=0038 DX=0000 SP=01BE  BP=0000 SI=0000 DI=0000
DS=1696 ES=1696 SS=1696 CS=1696 IP=0100  NV UP EI PL NZ NA PO NC
1696:0100 FD            STD
-d
1696:0100  FD A1 10 00 00 31 00 11-00 0F 00 00 00 7F 00 00   .....1.........
1696:0110  7F 00 00 7F 00 00 7F 00-00 7F 00 00 7F 00 00 7F   ...............
1696:0120  00 00 7F 00 00 7F 00 00-7F 00 00 7F 00 00 7F 00   ...............
1696:0130  00 7F 00 00 7F FF FF FF-04 7D 05 8B 04 A3 E8 1B   .........}.....
1696:0140  A1 EC 1B 39 05 7E 05 8B-05 A3 EC 1B 8B 5E E0 A1   ...9.........
1696:0150  EA 1B 39 07 7D 05 8B 07-A3 EA 1B 8B 5E E2 39 17   ..9.}.......9.
1696:0160  7E 02 8B 17 83 C6 02 83-C7 02 83 46 E0 02 83 46   ..........F...F
1696:0170  E2 02 E2 C0 89 16 EE 1B-A1 EC 1B 2B 06 E8 1B 99   ...........+....
```

有了前几次的经验后，我们可以很轻易地得知，此文件的长度记载在寄存器 CX 里，即为十进制的 56，于是把图形数据的范围框住，以利观察。

接着观察文件前部，FD 为识别符，3100 为全长减去前 7 个字节，11 00 为宽，0F 00 为高，截至目前为止和 BASIC 格式全部符合，一眼就看出这是一张 17×15 大小的图形。

多观察几个 MAP 图形文件后将会发现：在文件中记载段地址及偏移地址的第 1 个(从 0 开始算）到第 4 个字节，所有的单色 MAP 文件所记载的值都一样，同为 A1 10 00 00，难道所有的图形的段地址及偏移地址都一样？当然不是。这个问题牵扯到 BSAVE／BLOAD 在处理图形文件的一个现象：

虽然这 4 个字节所记载的是系统(指 BASIC)当时储存这张图形时的段地址及偏移地址，然而在 BLOAD 时，这些地址的值并不会影响到图形的载入和显示；也就是说，这 4 个字节可以是任何值而不一定是地址，所以这 4 个字节就可以供一些绘图软件作为一些重要的识别信息之用．Rambow PAINT 就是其中一个例子．Rabow PAINT 以第 1 个字节来识别图形的单彩色，A1 为单色 MAP 文件而 B1 则为彩色 MAP，另 3 个字节则固定为 10 00 00。

在此我们依旧将 MAP 文件格式整理如下：

表 3-4　MAP 图形文件的格式

起始字节	长度(Byte)	描　　述
0	1	文件识别记号，恒为 FD
1	1	色彩识别记号：A1 为单色，B1 为彩色
2	3	保留，恒为 10 00 00
5	2	图形数据全长
7	2	图形的宽度（以像素为单位）
9	2	图形的高度（以扫描线为单位）
11	全长-4	图形数据

既然 MAP 文件格式和 BASIC 兼容，那么读取 MAP 文件时，LOAD_BAS.C 便可以完全适用。请参考下列源程序：

```
/ * LOAD_MAP.C * /

#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>

void load_map(int, int, char *, int, int);

main()
{
     int graph_driver=HERCMONO, graph_mode=HERCMONOHI;

     initgraph(&graph_driver, &graph_mode, "");
     load_map(0,0,"demo.map",0,COPY_PUT);
     getch();
     closegraph();
}

void load_map(int x,int y,char *filename,int page,int status)
{
          int *buf;
          unsigned size;
          FILE *fp;
          int i;

          if((fp = fopen(filename, "rb"))==NULL) {
               closegraph();
               puts("File reading error!");
```

```
        exit(1);
    }

    for(i=0; i<7; i++)
        fgetc(fp);
    size = filelength(fileno(fp));
    buf = (int *) malloc(size);
    if(buf= =NULL) {
        closegraph();
        puts("Memory is not enough!");
        exit(1);
    }

    fread(buf, 1, size, fp);
    fclose(fp);
    buf[0]--;
    buf[1]--;
    setactivepage(page);
    putimage(x,y,buf,status);
    free(buf);
}
```

图 3-4　程序 LOAD_MAP.C 的执行结果

```
        exit(1);
    }

    for(i=0; i<7; i++)
        fgetc(fp);
    size = filelength(fileno(fp));
    buf = (int *) malloc(size);
    if(buf==NULL) {
        closegraph();
        puts("Memory is not enough!");
        exit(1);
    }

    fread(buf, 1, size, fp);
    fclose(fp);
    buf[0]--;
    buf[1]--;
    setactivepage(page);
    putimage(x,y,buf,status);
    free(buf);
}
```

图 3-4　程序 LOAD_MAP.C 的执行结果

3.2.5 BPC 文件格式的读取（绘图大师）

绘图大师所截取下来的图形文件，并没有强制地规定扩展名是什么，但其格式和变影 72 的 PUT 文件一模一样，如果绘图时将文件的扩展名命名成为 PUT，则这个图形文件便可直接在变影 72 中直接读出无须经过任何转换，当然在 3.2.3 节所介绍的程序也可以使用。

绘图大师的磁盘上另外附了一个简易的图形文件格式转换程序，可将绘制的图形转换成和 BASIC 兼容的 BPC 格式，和与 Turbo C 相容的 TPC 格式。本节将介绍 BPC 的格式，而在下一小节介绍 TPC 格式。

为了解 BPC 的格式，我们仍用 DEBUG 程序将一张 BPC 格式的图形内容列表如下：

```
-r
AX=0000 BX=0000 CX=1CF3 DX=0000 SP=01BE BP=0000 SI=0000 DI=0000
DS=1696 ES=1696 SS=1696 CS=1696 IP=1696 NV UP EI PL NZ NA PO NC
1696:0100 FD           STD
-d
1696:0100  FD 00 00 00 00  EC  1C  89-01  94  00  FF  FF  FF  FF  FF    ................
1696:0110  FF  FF FF FF FF  FF  FF  FF-FF FF  FF FF  FF  FF  FF  FF    ................
1696:0120  FF  FF FF FF FF  FF  FF  FF-FF FF  FF FF  FF  FF  FF  FF    ................
1696:0130  FF  FF FF FF FF  FF  FF  FF-FF FF  FF FF  80  FF  FF  FF    ................
1696:0140  FF  FF FF FF FF  FF  FF  FF-FF FF  FF FF  FF  FF  FF  FF    ................
1696:0150  FF  FF FF FF FF  FF  FF  FF-FF FF  FF FF  FF  FF  FF  FF    ................
1696:0160  FF  FF FF FF FF  FF  FF  FF-FF FF  FF FF  FF  FF  80  FF    ................
1696:0170  FF  FF FF FF FF  FF  FF  FF-FF FF  FF FF  FF  FF  FF  FF    ................
```

由 CX 的值可以知道，这是一个大张的图形，因此在这里我们只能看到图形文件的前端。因为本格式是设计成和 BASIC 的 BSAVE 文件相似的，所以，除了段地址及偏移地址（第 1 个 byte 到第 4 个 byte）全部填零外，其余部分完全和 BSAVE 文件一致。也就是说，读取 MAP 和 BASIC 图形文件的 C 程序可以直接拿来读取 BPC 图形文件。虽然这三种格式的前五个字节都不太一样，但是这 5 个字节的值不会影响图形文件的读取，所以程序码大致相同。

以下就是 BPC 文件的读取程序 LOAD__BPC.C：

```
/ * LOAD__BPC.C * /

#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>

void load__bpc(int, int, char *, int, int);
```

```
main()
{
     int graph__driver=HERCMONO, graph__mode=HERCMONOHI;

     initgraph(&graph__driver, &graph__mode, "");
     load__bpc(0,0,"demo.bpc",0,COPY__PUT);
     getch();
     closegraph();
}

void load__bpc(int x, int y, char *filename, int page, int status)
{
     int *buff;
     unsigned size;
     FILE *fp;
     int i;

     if((fp = fopen(filename, "rb"))==NULL) {
         closegraph();
         puts("File reading error!");
         exit(1);
     }
     for(i=0; i<7; i++)
         fgetc(fp);
     size = filelength(fileno(fp));
     buff = (int *) malloc(size);
     if(buff==NULL) {
         closegraph();
         puts("Memory is not enough!");
         exit(1);
     }
     fread(buff, 1, size, fp);
     fclose(fp);
     buff[0]--;
     buff[1]--;
     setactivepage(page);
     putimage(x,y,buff,status);
     free(buff);
}
```

有一点必须提醒读者注意的：假如你使用的 CONVERT.EXE 为 1.1 版的话，在转换文件时，请确定该文件是否真为绘图大师所绘下来的图形文件，而且你只能转换一次。因为 CONVERT.EXE 并不事先检查源文件是否真为图形文件，或是否已经转换过了，如果你将文件一再地转换，将会发现文件前端会不断地加上文件头，当然你的图形也会被破

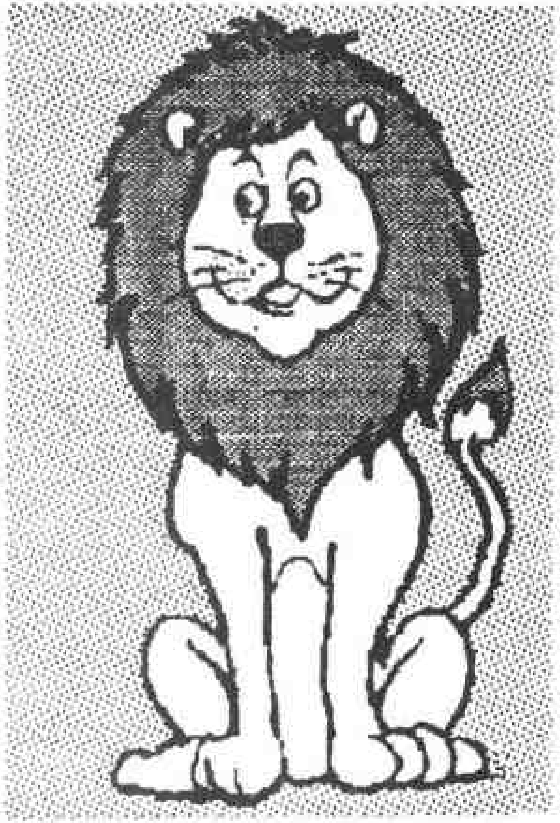
坏了。同时你也可以试着将任意的可执行文件交给 CONVERT.EXE 转换，它依然告诉你 OK！不过，所得到的文件当然不是我们想要的。

图 3-5　程序 LOAD_BPC.C 的执行结果

3.2.6　TPC 文件格式的读取（绘图大师）

绘图大师所提供的另一种格式为 TPC，这个格式和 3.2.1 节所介绍的用 getimage 所取得的图形格式一模一样，前面 4 个 bytes 分别代表图形的宽度减 1 及高度减 1，因此，可以直接读入使用而无须在程序码中再将宽高各减 1。这是读取 TPC 文件和读取 PUT 文件最大的不同点，也是唯一的不同点。请看程序列表，并请注意：这个程序和 LOAD_PUT.C 程序比较起来，少了 buf[0]--; 及 buf[1]--; 两行：

```
/* LOAD_TPC.C */

#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>

void load_tpc(int, int, char *, int, int);

main()
{
```



坏了．同时你也可以试着将任意的可执行文件交给 CONVERT.EXE 转换，它依然告诉你 OK！不过，所得到的文件当然不是我们想要的．

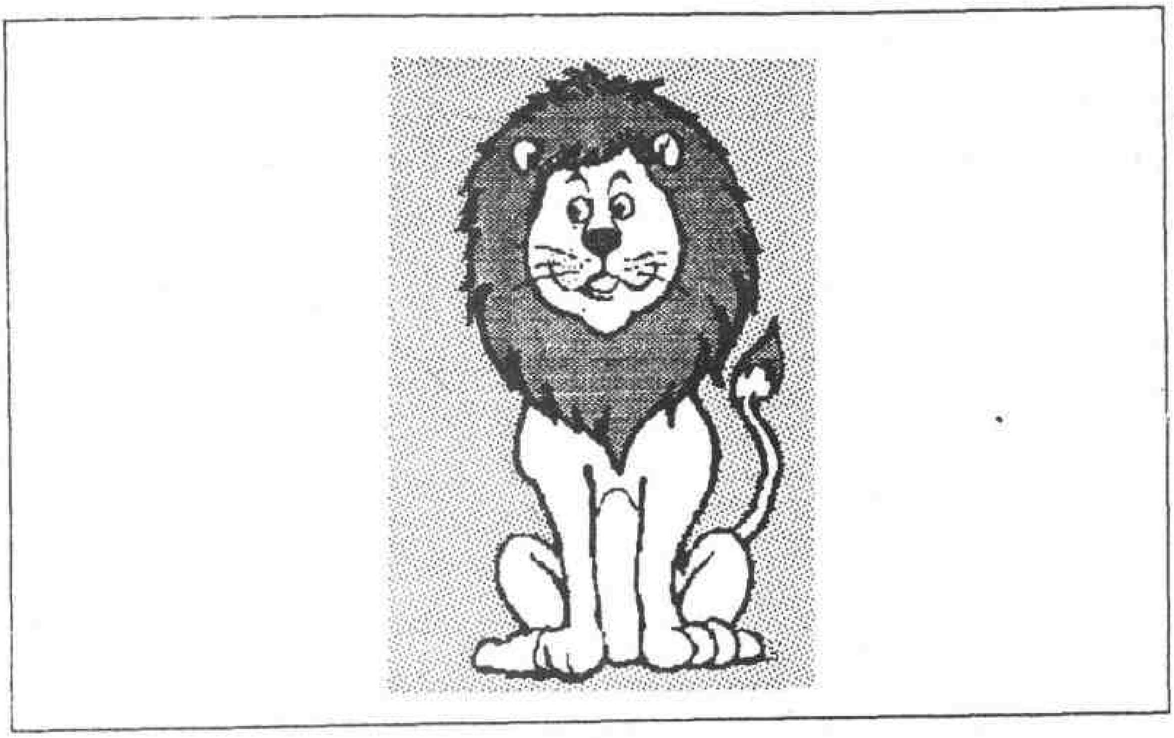


图 3-5　程序 LOAD__BPC.C 的执行结果

3.2.6　TPC 文件格式的读取（绘图大师）

绘图大师所提供的另一种格式为 TPC．这个格式和 3.2.1 节所介绍的用 getimage 所取得的图形格式一模一样，前面 4 个 bytes 分别代表图形的宽度减 1 及高度减 1，因此，可以直接读入使用而无须在程序码中再将宽高各减 1．这是读取 TPC 文件和读取 PUT 文件最大的不同点，也是唯一的不同点．请看程序列表，并请注意：这个程序和 LOAD__PUT.C 程序比较起来，少了 buf[0]--；及 buf[1]--；两行：

```
/* LOAD__TPC.C */

#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>

void load__tpc(int, int, char *, int, int);

main()
{
```

```
	int graph_driver=HERCMONO, graph_mode=HERCMONOHI;

	initgraph(&graph_driver, &graph_mode, "");
	load_tpc(0,0,"demo.tpc",0,COPY_PUT);
	getch();
	closegraph();
}

void load_tpc(int x,int y,char *filename,int page,int status)
{
	int *buf;
	unsigned size;
	FILE *fp;
	int i;
	if((fp = fopen(filename, "rb"))==NULL) {
		closegraph();
		puts("File reading error!");
		exit(1);
	}
	size = filelength(fileno(fp));
	buf = (int *) malloc(size);
	if(buf==NULL) {
		closegraph();
		puts("Memory is not enough!");
		exit(1);
	}
	fread(buf, 1, size, fp);
	fclose(fp);
	setactivepage(page);
	putimage(x,y,buf,status);
	free(buf);
}
void load_bpc(int x,int y,char *filename,int page,int status)
{
	int *buf;
	unsigned size;
	FILE *fp;
	int i;
	if((fp = fopen(filename, "rb"))==NULL) {
		closegraph();
		puts("File reading error!");
		exit(1);
	}
	for(i=0; i<7; i++)
```

```
        fgetc(fp);
    size = filelength(fileno(fp));
    buf = (int *) malloc(size);
    if(buf==NULL) {
        closegraph();
        puts("Memory is not enough!");
        exit(1);
    }
    fread(buf, 1, size, fp);
    fclose(fp);
    buf[0]--;
    buf[1]--;
    setactivepage(page);
    putimage(x,y,buf,status);
    free(buf);
}
```

图 3-6 程序 LOAD__TPC.C 的执行结果

3.2.7 PCX 文件格式的读取（PC Paint Brush、达文西二代等）

前面几节所介绍的文件格式都是属于区块图形文件，结构简单且未经压缩，因此可轻易地读取使用。然而正因为它没有压缩处理，所以并不适合储存大张的图形，以免浪费磁

Terminator II

```
        fgetc(fp);
    size = filelength(fileno(fp));
    buf = (int *) malloc(size);
    if(buf==NULL) {
        closegraph();
        puts("Memory is not enough!");
        exit(1);
    }
    fread(buf, 1, size, fp);
    fclose(fp);
    buf[0]--;
    buf[1]--;
    setactivepage(page);
    putimage(x,y,buf,status);
    free(buf);
}
```

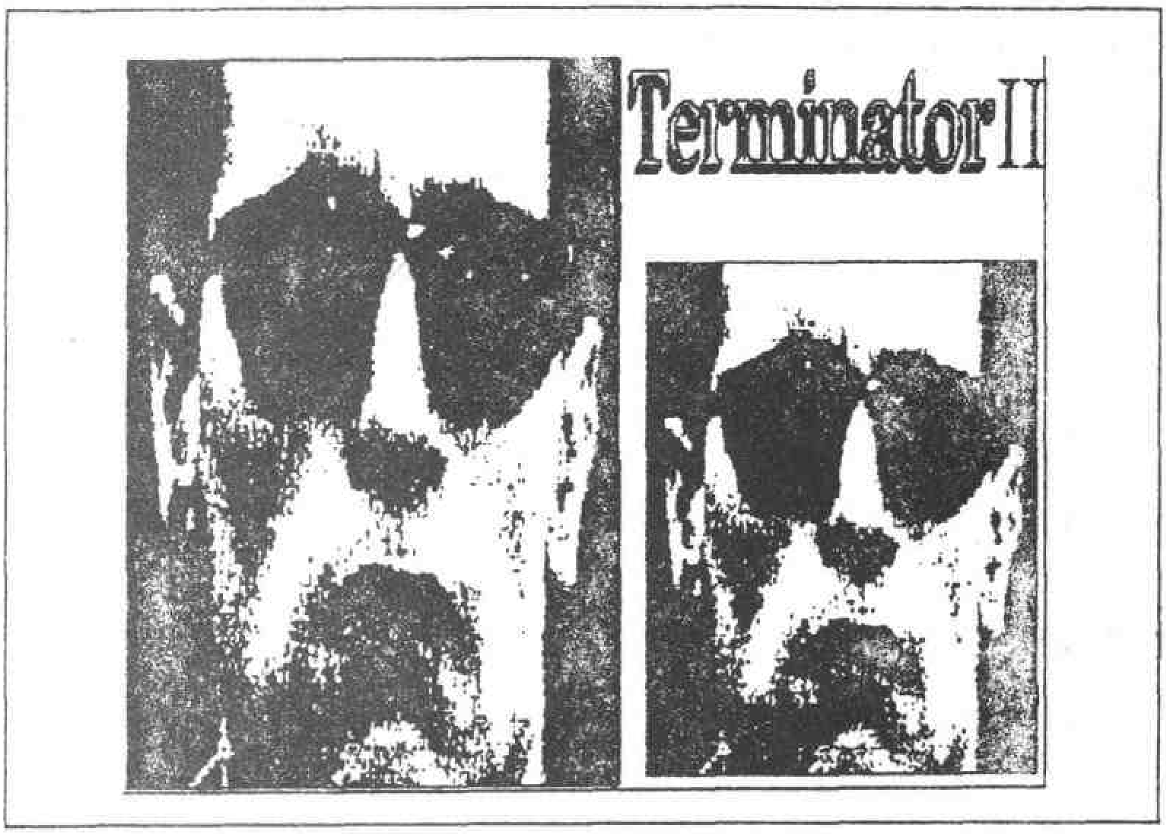


图 3-6 程序 LOAD__TPC.C 的执行结果

3.2.7 PCX 文件格式的读取（PC Paint Brush、达文西二代等）

前面几节所介绍的文件格式都是属于区块图形文件，结构简单且未经压缩，因此可轻易地读取使用。然而正因为它没有压缩处理，所以并不适合储存大张的图形，以免浪费磁

盘空间，适当地压缩是必要的。本节所要介绍的就是经过压缩的图形文件 PCX。

PCX 文件格式在 PC 上的应用十分普遍，许多国内常见的绘图软件，如 PC PaintBrush，GEM，Z-Soft Soft-type，Macstro，TOPAS Show，变影 72，达文西二代，优秀大师……等等都是采用 PCX 文件。

这种文件格式在正式储存图形数据之前，在文件的最前端有一个长度固定为 128 字节的文件头（Flie Header)、记载着有关 PCX 文件的重要信息，其详细内容如表 3-5 所示:

表 3-5 PCX 图形文件的格式

起始字节	长度(Byte)	描　　　　述
0	1	制造商，通常为 0x00
1	1	版本代号
2	1	编码代号，通常为 1
3	1	每个像素以多少 bit 表示
4	4	图的左上角坐标 (Xmin,Ymin)
8	4	图的右下角坐标 (Xmax,Ymax)
12	2	水平分辨率
14	2	垂直分辨率
16	48	色彩调色盘
64	1	保留
65	1	色彩平面的数目
66	2	每条扫描线有几个 byte
68	2	调色盘的形式，1 为彩色或黑白，2 为灰阶
70	58	填满文件头

以上的文件头在 C 语言里可以用如下的结构表示:

```
typedef struct {
    char manufacturer;
    cahr version;
    char encoding;
    char bits__per__pixel;
    int xmin, ymin;
    int xmax, ymax;
    int hres;
    int vres;
    char palette[48];
    cahr reserved;
    char num__of__planes:
```

```
    int bits__per__line;
    int type__of__palette;
    char filler[58]
}PCX__HEAD;
```

这个 128 字节的文件头，如果我们只考虑单色的 PCX 文件的话，那么我们真正感兴趣的就只有 bits__pex__pixel，xmin，ymin，xmax，ymax 5 项罢了．检查 bits__pex__pixel 我们可以得知此图形文件是否为单色。

bits__pex__pixel＝1 时为单色图形文件

而(xmin,ymin)及(xmax,ymax)则可以用来求得图形的大小。

接下来介绍 PCX 的压缩方法．PCX 文件的压缩方法是属于 Run-Length 方式．所谓 Run-Length 方式，就是图形数据的重复次数乃是取决于前面一个计数器而定．然而从一连串的位中如何去分辨那一个字节是数据本身而那一个字节是计数器呢？PCX 用了如下的识别方法：

1. 假设从一连串的字节中取得一个字节为：

   10001101

2. 判断其最左边的两个位，如果同时为1，则这个位为计数器，重复次数为最右边6个位的值，而下一个字节组是图形数据。

3. 如果最左边的两个位不同时为1，则这个字节本身就是图形数据，重复次数为1。

在此举个实例进行分析：

未展开前的字串为：

C2　FF　C5　00　OF　05　C1　FF

上列字串的计数器分别为：

C2＝ 11 00 00 10 重复次数＝2

C5＝ 11 00 01 01 重复次数＝5

C1＝ 11 00 00 01 重复次数＝1

所以展开后的字串应为：

FF　FF　00　00　00　00　00　OF　05　FF

将以上的方法写成 C 程序段如下所示：

```
/ * cnt 表示重复次数 * /
/ * picdate 表示图形数据 * /
  cnt＝1;
  picdata＝fgetc(fp); / * 读取 1 个字节 * /
  if(0xc0＝＝(0xc0 &picdata){ / * 判断最左边 2 个比特是否全为 1 * /
    cnt＝0x3f &picdata; / * 是，则 cnt＝最右边 6 比特之值 * /
    picdata＝fgetc(fp); / * 再读取 1 个字节为重复数据 * /
}
```

接着则是完整程序：

```c
/* LOAD_PCX.C */

#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>

void load_pcx(int, int, char *, int, int);

main()
{
      int graph_driver=HERCMONO, graph_mode=HERCMONOHI;

      initgraph(&graph_driver, &graph_mode, "");
      load_pcx(0,0,"demo.pcx",0,COPY_PUT);
      getch();
      closegraph();
}

void load_pcx(int x,int y,char *filename,int page,int status)
{
      typedef struct {
          char manufacturer;
          char version;
          char encoding;
          char bits_per_pixel;
          int  xmin, ymin;
          int  xmax, ymax;
          int  hres;
          int  vres;
          char palette[48];
          char reserved;
          char nplanes;
          int bits_per_line;
          int  palette type;
          char filler[58];
                } PCX_HEAD;
      FILE *fp;
      int i;
      char picdata;
      unsigned xlength, ylength, xlength_b;
      unsigned size, cnt=0, total=0;
      PCX_HEAD_header;
      char *buf;
```

```
    int  * s;

    if((fp = fopen(filename, "rb"))==NULL) {
        closegraph();
        puts("File Reading Error!!");
        exit(1);
    }
    fread((char  * ) &header, 1, sizeof(PCX_HEAD), fp);
    xlength  =  header.xmax - header.xmin + 1 ;
    ylength  =  header.ymax - header.ymin + 1;
    if((xlength>720)||(ylength>348)) {
        closegraph();
        puts("Picture is too large!");
        exit(1);
    }
    size = imagesize( header.xmin, header.ymin,
                          header.xmax, header.ymax);
    buf = (char  * ) malloc(size);
    s = (int  * ) buf;
    buf += 4;   / *  Reserve the space for length of x and y  * /
    if(xlength%8)
        xlength_b = (xlength >> 3) + 1;
    else
        xlength_b = xlength >> 3;

    while(total<xlength_b*ylength)   {
        cnt = 1;
        picdata = fgetc(fp);
        if(0xc0==(0xc0 & picdata)) {
            cnt = 0x3f & picdata;
            picdata = fgetc(fp);
        }
        for(i=0; i<cnt; i++)
            * (buf++) = picdata;
        total += cnt;
    }
    fclose(fp);

    setactivepage(page);
    s[0] = xlength - 4;
    s[1] = ylength - 1;
    putimage(x, y, buf, status);
    free(s);
}
```

程序说明:

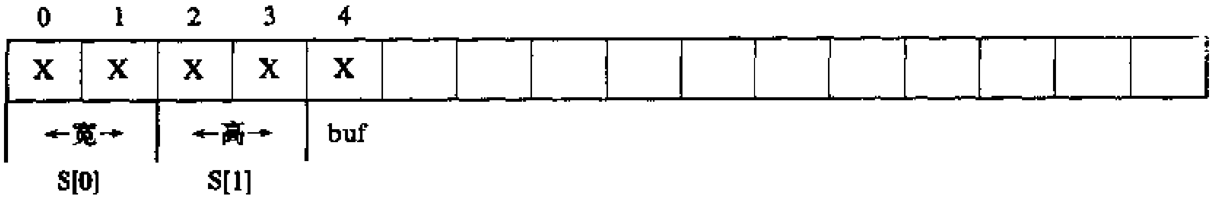
由于上列的程序可能较为复杂，因此在此列出其变量的意义及程序流程，以利读者了解本程序。

一、变量的意义:

picdata　：储存1个字节的图形数据。
xlength　：以像素为单位的图形宽度。
ylength　：以扫描线为单位的图形高度。
xlength_b　：以字节为单位的图形宽度。
size　：图形的大小，以字节为单位。
cnt　：重复次数。
total　：经展开后的字节总数。
header　：文件头的结构变数。
buf　：储存展开后图形的连续内存区块指针。
s　：指向buf的整数指针。

二、程序流程及说明:

47-51　打开文件
52　读取文件头
53-54　计算图形的宽、高
60-62　配置足够的内存空间，并以 buf 指针指向起始地址。
64-68　整数指针s指向buf所指的地址。s指针永远指向内存区块的开头，而buf指针随着数据的存入而指向下一个要存入数据的地址。如下图所示:



buf+=4; 数据从第 4 个字节开始存起。
算出图宽共为多少位数。

70-80　展开，并将压缩后的数据存入 buf 所指的地址。
81　打开文件。
84-85　存入图宽及图高，即 S[0]及 S[1]
86　显示此图。
87　释放存储空间。

3.2.8 PMT 文件格式的读取（变影 72）

PMT 是变影 72 的全页图形文件，其压缩方式也是利用 Run-Length 的方法。然而同样是 Run-Length 的方法，却表示不同。

单色的 PMT 文件展开后，其全图的大小为 1728 pixels×1100 lines；换句话说，图高为 1100 扫描线，而每条扫描线由 216 个字节来表示。

图 3-7 程序 LOAD_PCX.C 的执行结果

这个文件格式并没有一般图形格式所拥有的文件头（File Header），而是直接由四段连续的描述字串所组成，而每段描述字串之前都有 2 个 byte 来表示紧接着的描述字符的长度，虽然压缩后的描述字串长度不一，但是每段所展开后的长度都是 59400 字节，而全部的图形数据长度应为 59000×4 字节，当然也等于 216×1100，根据以上的分析，PMT 的文件结构应该如以下的图形所示：

描述字串长度	第 1 段描述字串	描述字串长度	第 2 段描述字串	描述字串长度	第 3 段描述字串	描述字串长度	第 4 段描述字串

然而，其描述字串的内容应该如何表示呢？分析起来共有如下两种情况：

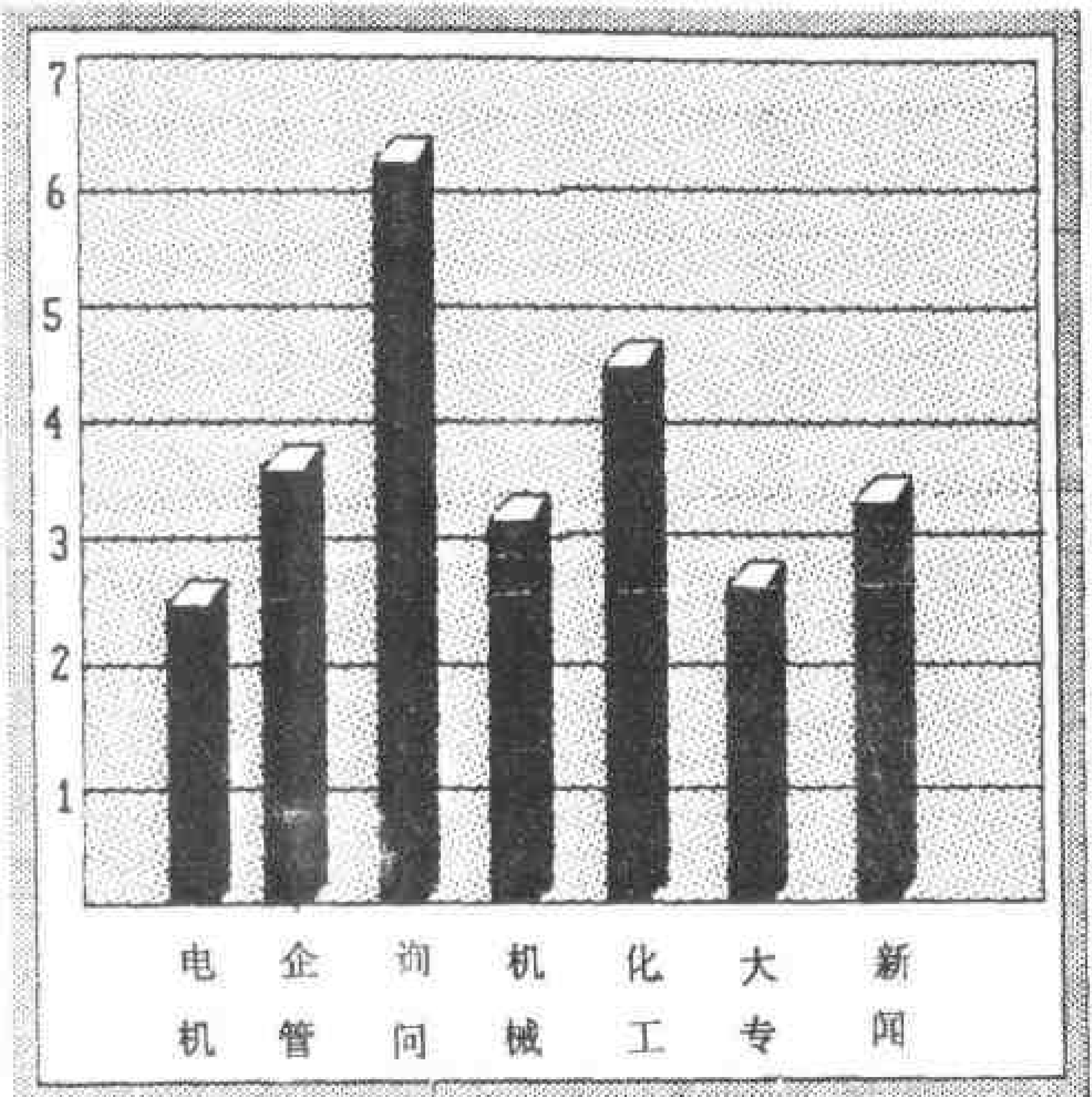
第一种情况：FF 50

如果读取到的字节的最左位为 1，那么下一个字节重复次数即为本字节的最右 7 个位的值。就如以上例子而言，即表示将 50 重复 127 次。

第二种情况：04 FF A0 84 53

如果读取到的字节的最左边位为 0，则此位的值即为紧接着不重复字节的个数，如本例即直接填入 FF A0 84 53 四个字节即可。

在此举个例子，让各位读者能更清楚此种压缩方式。假设有如下的描述字串：



7
6
5
4
3
2
1
电机
企管
询问
机械
化工
大专
新闻

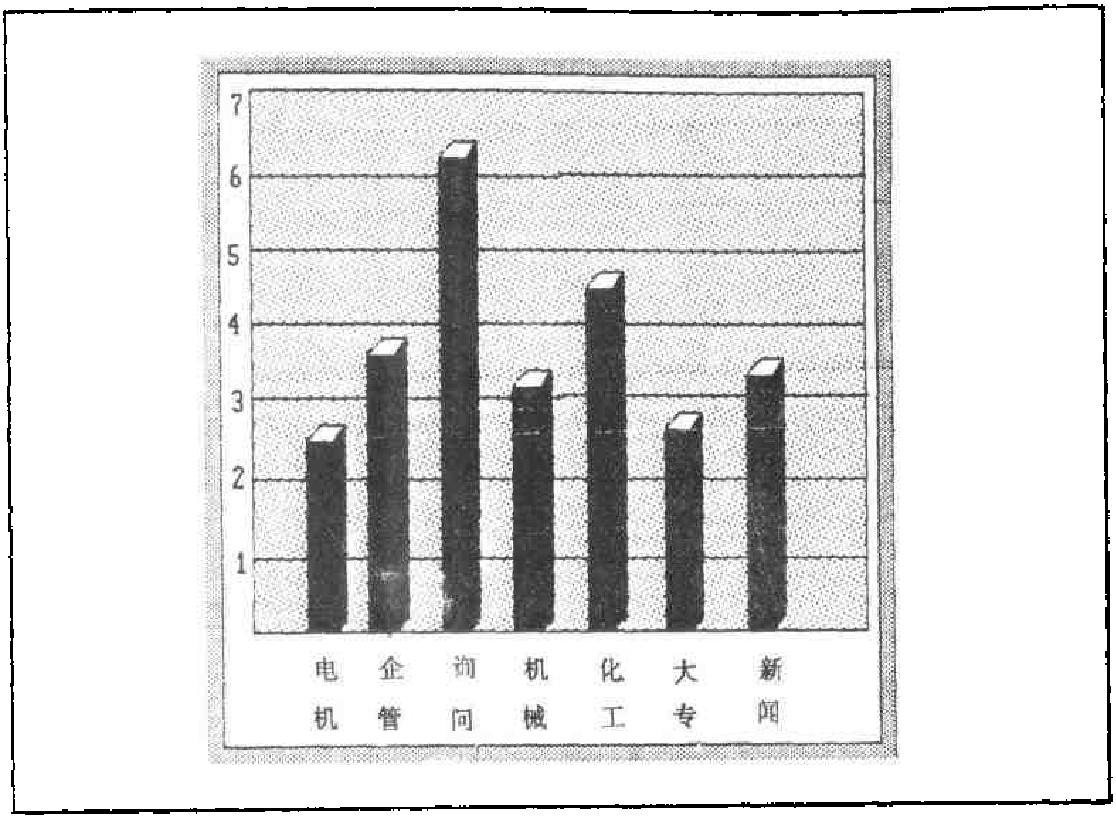


图 3-7 程序 LOAD_PCX.C 的执行结果

这个文件格式并没有一般图形格式所拥有的文件头（File Header），而是直接由四段连续的描述字串所组成，而每段描述字串之前都有 2 个 byte 来表示紧接着的描述字符的长度，虽然压缩后的描述字串长度不一，但是每段所展开后的长度都是 59400 字节，而全部的图形数据长度应为 59000×4 字节，当然也等于 216×1100，根据以上的分析，PMT 的文件结构应该如以下的图形所示：

描述字串长度	第 1 段描述字串	描述字串长度	第 2 段描述字串	描述字串长度	第 3 段描述字串	描述字串长度	第 4 段描述字串

然而，其描述字串的内容应该如何表示呢？分析起来共有如下两种情况：

第一种情况：FF 50

如果读取到的字节的最左位为 1，那么下一个字节重复次数即为本字节的最右 7 个位的值。就如以上例子而言，即表示将 50 重复 127 次。

第二种情况：04 FF A0 84 53

如果读取到的字节的最左边位为 0，则此位的值即为紧接着不重复字节的个数。如本例即直接填入 FF A0 84 53 四个字节即可。

在此举个例子，让各位读者能更清楚此种压缩方式。假设有如下的描述字串：

84 F8 03 F5 64 32 85 00 85 FF 02 74 58

我们特别将计数字节标出，以供读者检查核对。这个描述字串展开后如下所示：

F8 F8 F8 F5 64 32 00 00 00 00 FF FF FF FF FF 74 58

特别用线标示在一起，就是由同一个计数字节所解码出来的字节。以下即为读取 PMT 格式程序 LOAD_PMT.C 的列表：

```
/ * LOAD_PMT.C * /

#include <graphics.h>
#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>

void load_pmt(char *, int, int);

main()
{
    int graph_driver=HERCMONO, graph_mode=HERCMONOHI;

    initgraph(&graph_driver, &graph_mode, "");
    load_pmt("demo.pmt", 0, COPY_PUT);

    getch();
    closegraph();
}

void load_pmt(char *filename, int page, int status)
{
    FILE *fp;
    unsigned int counter, number, cnt, x, y;
    register int i;
    char data;
    char *buf;
    unsigned size;

    size = imagesize(0,0,getmaxx(), getmaxy());
    buf = (char *) malloc(size);
    if(buf==NULL) {
        closegraph();
        puts("Memory is not enough");
        exit(1);
    }
    buf[0] = 719 % 256;    / * The width of picture * /
    buf[1] = 719 / 256;
```

```
buf[2] = 347 % 256;    /* The height of picture */
buf[3] = 347 / 256;
if((fp=fopen(filename, "rb"))==NULL) {
    closegraph();
    puts("File reading error!");
    exit(1);
}
fread(&number, sizeof(int), 1, fp);
counter = 0;
y = 0;
while(1)      {
    x = 0;
    while(x<216)              {
        data = fgetc(fp);
        counter ++;
        cnt = data & 0x7f;

        if( !(data>>7) )
        {   /* if MSB of data is clear */
            for(i=0; i<cnt; i++)      {
                if(x<90)
                    buf[y*90+x+4]=fgetc(fp)&0xff;
                else
                    fgetc(fp);
                x ++;
                if(x>=216)      {
                    x = 0;
                    y ++;
                    if(y==348) goto end;
                }
            }
            counter += cnt;
        }

        else              {   /* if MSB of data is set */
            data = fgetc(fp);
            counter ++;
            for(i=0; i<cnt; i++)      {
                if(x<90)
                    buf[y*90+x+4] = data & 0xff;
                x ++;
                if(x>=216)      {
                    x = 0;
                    y ++;
```

```
83:                         if(y = = 348) goto end;
84:                     }
85:                 }
86:                 }
87:                 if(counter> =number)          {
88:                       fread(&number, sizeof(int), 1, fp);
89:                 counter = 0;
90:                 }
91:           }
92:     }
93:     end:;
94:     fclose(fp);
95:     setactivepage(page);
96:     putimage(0,0,buf,status);
97:     free(buf);
98:}
```

程序说明:

一、这个程序所使用到的重要变量说明如下:

Counter ：记录目前已展开完毕的描述字串字节数。

number ：记录描述字串的长度。

cnt ：记录重复次数或不重复字节数。

x ：指向目前正在展开的该条扫描线的第 X 字节。

y ：指向目前正在展开的扫描线。

data ：储存被重复的图形数据。

buf ：储存展开后的图形数据。因为PMT格式的宽高都大于一个屏幕，而我们只需要储存屏幕大小，所以储存的范围应为 0<X<90, 0<Y<348。因为展开完毕后想使用 putimage 将图形显示在屏幕上，因此 buf 的格式必须符合 getimage 后的格式，也即 buf 的前 4 个字节分别记录整个屏幕的宽减 1 及高减 1，即 719 和 347。

二、LOAD_PMT.C 的流程

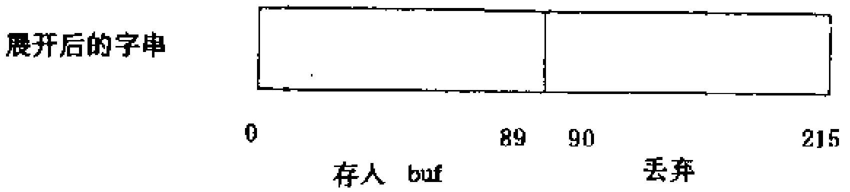
41−45 启动文件。

30−32 配置整页屏幕大小的内存，由 buf 指向起始字节位置。

37−40 buf[0]及 buf[1]分别填入 719 的低字节及高字节，即屏幕的宽减 1。
buf[2]及 buf[3]分别填入 347 的低字节及高字节，即屏幕的高减 1。

46 读取描述字串的长度放入 number。

51−92 对本段描述字串进行展开，因为PMT的宽度的216字节，所以需做如下处理:



当一段扫描字串都已经展开完毕后再回到第五步骤，直到展开到扫描线第347条为止.

94 打开文件.

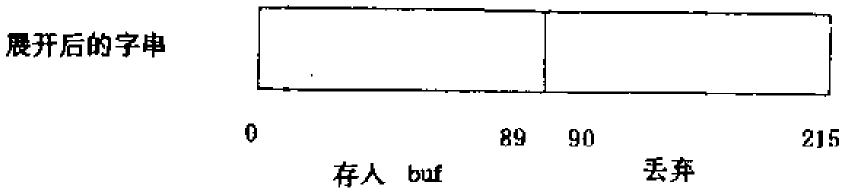
95 显示此图.

96 释放内存.

图 3-8 程序 LOAD_PMT.C 的执行结果

3.2.9 DVC 文件格式的读取（达文西）

单色板的 DVC 文件所用的图形工作页宽为 640 个像素，高为 500 条扫描线，一样是属于 Run-Length 的压缩方式，其文件结构如下:



当一段扫描字串都已经展开 完毕后再回到第五步骤，直到展开到扫描线第 347 条为止.

94 打开文件.

95 显示此图.

96 释放内存.

图 3-8 程序 LOAD__PMT.C 的执行结果

3.2.9 DVC 文件格式的读取（达文西）

单色板的 DVC 文件所用的图形工作页宽为 640 个像素，高为 500 条扫描线，一样是属于 Run-Length 的压缩方式，其文件结构如下:

表 3-6 DVC 图形文件的格式

起始字节	长 度	描 述
0	8	文件识别码
8	2	图宽
10	2	图高
12	全长-12	500 组图形描述字串

我们用 DEBUG 显示出一个 DVC 格式文件的前部分来对照一下：

```
-d
64AF:0100  0B 48 73 75 4D 61 70 00-50 00 F4 01 04 01 01 05   .HsuMap.P......P
64AF:0110  FF 04 01 01 50 FF 04 01-01 50 FF 04 01 01 50 FF   .....P....P....P.
64AF:0120  04 01 01 50 FF 04 01 01-05 FF 04 01 01 50 FF 04   ...P....P....P...
64AF:0130  01 01 50 FF 04 01 01 50-FF 04 01 01 50 FF 04 01   ..P....P.....P...
64AF:0140  01 50 FF 0A 02 01 06 48-FF FC 00 03 05 FF 0B 02   .P....P....P....
64AF:0150  01 07 48 FF 80 00 00 0F-04 FF 0C 02 01 08 47 FF   ..H....H.....G..
64AF:0160  F8 02 AA A8 01 04 FF 0D-03 01 04 08 47 FF E0 03   ...........G....
64AF:0170  55 50 3F 03 FF 0C 03 01-04 07 47 FF 82 04 AA 0F   UP?........G.....
```

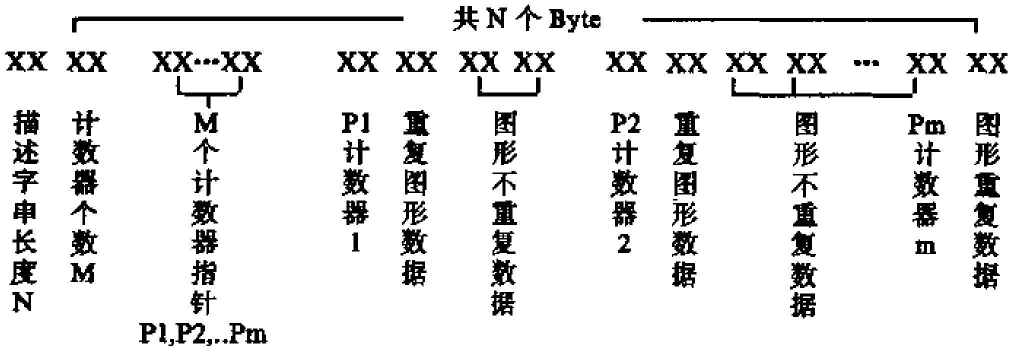
其中用底线画起来的部分的意义为：

0B 48 73 4D 61 70 00 为文件识别字串

50 00 等于 80 字节(即 640 pixel)，为展开后图形的宽度。

F4 01 等于十进制的 500，即 500 条扫描线，为展开后图形的高度。

紧接着便是以每条扫描线为压缩单位的 500 组压缩描述字串（因为有 500 条扫描线），描述字串的结构如下：



在描述字串结构中：

1. 第 1 个字节记录该描述字串的长度减 1（亦即不包含本身）。
2. 第二个字节记录的是描述字串中共有几个字节为计数器而非图形数据本身。
3. 第三个字节开始便指明了哪几个字节是计数器。除计数器本身及计数器的下一个重复字节外，图形描述字串中的其它字节即是不重复的图形数据。

举个例子，假设有如下的描述字串：

04　01　01　50　FF

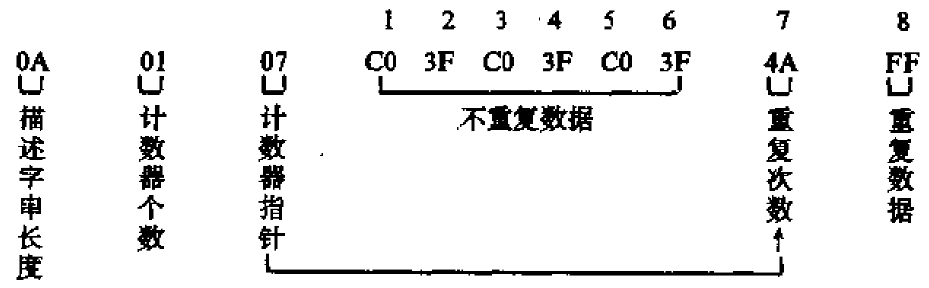
则 04 代表在 04 后面的描述字串长度为 4 个 byte，接着第 1 个 01 代表共有一个计数器，而计数器的位置为紧接在计数器指针后的第 01 个字节（50）。当然，跟在 50 后的 FF 就是将被重复 80 次的图形数据。

下一个例子，你可以指出哪几个字节是计数器吗？

				1	2	3	4	5	6	7
0A	02	03	06	FF	80	10	00	1F	3D	FF

答案是 10 和 3D。

再来看一个例子：



接下来请看 LOAD_DVC.C 的程序：

```
/ * LOAD_DVC.C * /

#include <graphics.h>
#include <stdio.h>
#include <alloc.h>

void load_dvc(char * int, int);

main()
{
      int graph_driver=HERCMONO, graph_mode=HERCMONOHI;

      initgraph(&graph_driver, &graph_mode, "");
      load_dvc("demo.dvc", 0, COPY_PUT);

      getch();
      closegraph();
}

```

```
void load_dvc(char * filename, int page, int status)
{
    FILE * fp;
    register int i;
    int str_length, counter_number, index;
    int counter_pointer[60], pointer = 1;
    int str_pointer = 0;
    int rept;
    char data;
    int x = 0, y = 0;
    char string[50];
    char * buf;
    unsigned size;

    size = imagesize(0,0,getmaxx(), getmaxy());
    buf = (char * ) malloc(size);
                            / * allocate memory of one page * /
    if(buf = = NULL) {
        closegraph();
        puts("Memory is not enough");
        exit(1);
    }
    buf[0] = 639 % 256;          / * The width of picture * /
    buf[1] = 639 / 256;
    buf[2] = 347 % 256;          / * The height of picture * /
    buf[3] = 347 / 256;

    if((fp = fopen(filename, "rb")) = = NULL)    {
        closegraph();
        puts("\aFile reading error!");
        exit(0);
    }

    for(i = 0; i < 12; i++)                  / * skip the file header * /
        fgetc(fp);
    for(y = 0; y < 348; y++) {
        x = 0;
        index = 0;
        pointer = 1;
        str_pointer = 0;
        str_length     = (int) fgetc(fp) & 0xff;
        counter_number = (int) fgetc(fp) & 0xff;
        for(i = 0; i < counter_number; i++)
        counter_pointer[i] = (int) fgetc(fp) & 0xff;
```

```
64:         counter__pointer[i] = -1;
65:         str__pointer = counter__number + 1;
66:
67:         while(str__pointer < str__length)
68:             if(counter__pointer[index] = = pointer)      {
69:                   index ++;
70:                   rept = (int) fgetc(fp) & 0xff;
71:                   rept = (rept > 80) ? 80 : rept;
72:                   data = fgetc(fp);
73:                   str pointer += 2;
74:                         pointer += 2;
75:         }
76:         else    {
77:                   rept = 1;
78:                   data = fgetc(fp);
79:                   str__pointer ++;
80:                   pointer ++;
81:             }
82:             for(i=x; i<x+rept; i++)
83:                   buf[y * 80+i+4] = data;
84:             x += rept;
85:         }
86:     }
87:     fclose(fp);
88:     setactivepage(page);
89:     putimage(0,0,buf,status);
90:     free(buf);
91:}
```

程序说明：

一、程序中所用到重要的变数说明如下：

buf	：	储存展开后的图形数据。
str__length	：	描述字串全长。
counter__number	：	计数器个数。
counter__pointer[ ]	：	储存指向各计数器的指针。
index	：	计数器指针数组中的下标。
printer	：	描述字串指针。从计数器指针后开始计算.
str__pointer	：	描述字串指针。记载目前已解码过的字串长度.

二、程序的流程如下：

34–35　配置整页的内存给 buf。

42–45　填入宽减 1 到 buf 的前 4 个字节。

47–51　打开文件.

53–54　丢弃前 12 个字节.

55–86　展开前 348 组描述字串（即 348 条扫描线）

87　　关闭文件.

89　　显示此图.

90　　释放内存.

图 3–9　程序 LOAD_DVC.C 的执行结果

3.3　文件格式的转换

正如本章第一节所说的，没有任何一套绘图软件能完全符合用户之需要，因此每一个用户可能会在各种不同的需求下使用到不同的绘图软件。如此一来，文件格式的转换程序便成了绘图软件相互间沟通不可缺少的工具了。

前面看到的图形文件格式转换程序，一般而言都是单纯的转换程序；也就是说，用户在转换之前，只能告诉程序所要转换的图形文件名称，以及要转换成的格式，在转换过程中根本看不到图形本身，更不用说做指定部分的图形转换了。有鉴于此，本节所要开发的文件格式转换程序，它将具备的特色如下：

1. 具有观看图形的功能.

达文西

图形文件

53–54　丢弃前 12 个字节.

55–86　展开前 348 组描述字串（即 348 条扫描线）

87　　关闭文件.

89　　显示此图.

90　　释放内存.

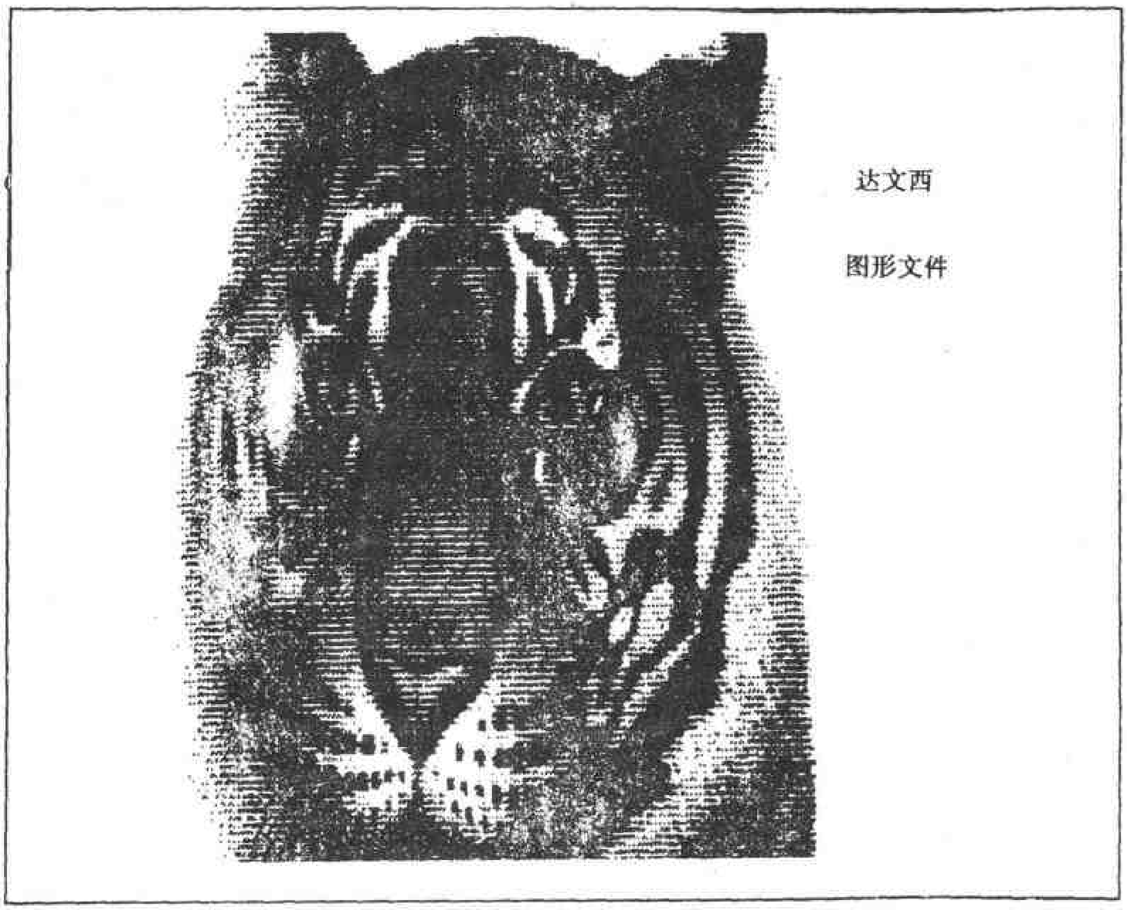


图 3–9　程序 LOAD_DVC.C 的执行结果

3.3　文件格式的转换

正如本章第一节所说的，没有任何一套绘图软件能完全符合用户之需要，因此每一个用户可能会在各种不同的需求下使用到不同的绘图软件. 如此一来，文件格式的转换程序便成了绘图软件相互间沟通不可缺少的工具了.

前面看到的图形文件格式转换程序，一般而言都是单纯的转换程序；也就是说，用户在转换之前，只能告诉程序所要转换的图形文件名称，以及要转换成的格式，在转换过程中根本看不到图形本身，更不用说做指定部分的图形转换了. 有鉴于此，本节所要开发的文件格式转换程序，它将具备的特色如下：

1. 具有观看图形的功能.

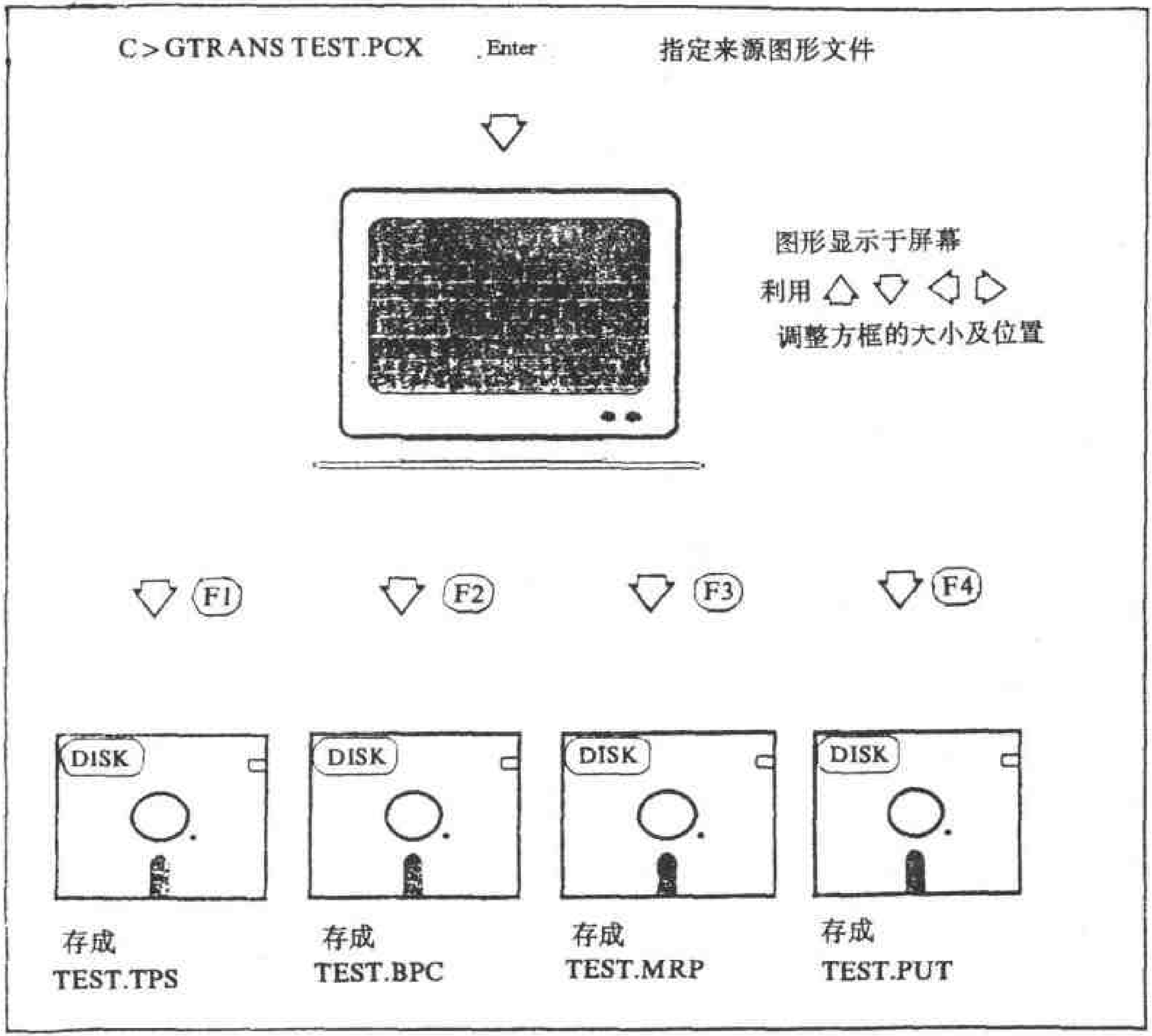
2. 可指定图形转换的区域，即做局部转换。

接着，我们明确地将整个转换流程图示如下:

图 3-10 图形转换程序 GTRANS.C 的流程图

以下我们更详细地将要求功能罗列于后:

1. 可输入七种图形文件格式，包括PCX，PMT，DVC，TPC，BPC，MAP，PUT。
2. 可输出四种图形文件格式，包括 TPC，BPC，MAP，PUT。
3. 利用上下左右箭头键调整要转换的图形位置及大小。
4. 可按+-键改变方框移动的速度。
5. 按 Esc 键可立即结束本程序。
6. 按 F1 键将方框所围起来的部分转换成 TPC 的格式。
7. 按 F2 键将方框所围起来的部分转换成 BPC 的格式。
8. 按 F3 键将方框所围起来的部分转换成 MAP 的格式。
9. 按 F4 键将方框所围起来的部分转换成 PUT 的格式。



C>GTRANS TEST.PCX
Enter
指定来源图形文件
图形显示于屏幕
利用
调整方框的大小及位置
F1
F2
F3
F4
DISK
DISK
DISK
DISK
存成
TEST.TPS
存成
TEST.BPC
存成
TEST.MRP
存成
TEST.PUT

2. 可指定图形转换的区域，即做局部转换。

接着，我们明确地将整个转换流程图示如下：

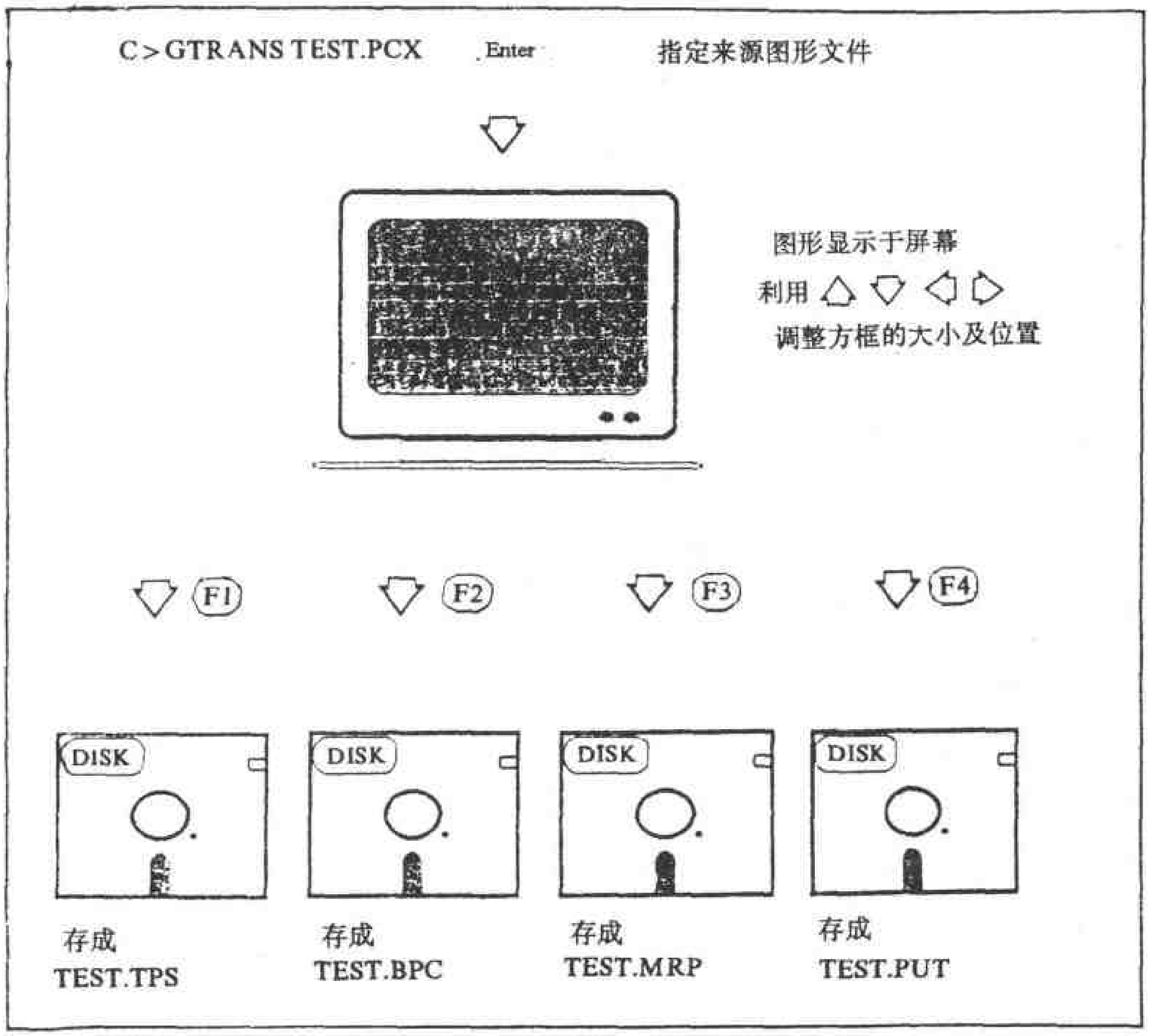


图 3-10 图形转换程序 GTRANS.C 的流程图

以下我们更详细地将要求功能罗列于后：

1. 可输入七种图形文件格式，包括PCX，PMT，DVC，TPC，BPC，MAP，PUT。
2. 可输出四种图形文件格式，包括 TPC，BPC，MAP，PUT。
3. 利用上下左右箭头键调整要转换的图形位置及大小。
4. 可按+-键改变方框移动的速度。
5. 按 Esc 键可立即结束本程序。
6. 按 F1 键将方框所围起来的部分转换成 TPC 的格式。
7. 按 F2 键将方框所围起来的部分转换成 BPC 的格式。
8. 按 F3 键将方框所围起来的部分转换成 MAP 的格式。
9. 按 F4 键将方框所围起来的部分转换成 PUT 的格式。

根据上述的定义及要求，接下来我们就将每一个制作的步骤分开来详细地讨论:

3.3.1 首先将指定的文件显示在屏幕上

依照前面的定义，我们的程序共有七种可能的格式输入，而判别用户输入的是何种格式，最简单的方法便是取得输入文件名的扩展名，然后做字串比较。我们将七种格式加以编号: PCX，PMT，DVC，TPC，BPC，MAP，PUT，分别为 1～7。如果比较符合就返回其相对应的值；如果全都不符合，那就表示不是本程序所支持的格式，返回-1。

主程序的写法如下:

```
char filename[13]
```

令来源图形文件的文件名为整体变量。

```
main(argc,argv)
int  argc;
char  * argv[];
```

取得命令行参数。

```
{
int graph__drvierr=HERCMONO,graph__mode=HERCOMNOHI;
int format;

if(argc<2)  {
      puts("Usage :GTRANS [graphics filename]");
      exit(1);
}
```

如果命令行参数的数目小于 2，表示未加参数，则输出使用说明。

```
strcpy(filename,argv[1]);
initgraph( &graph__driver,&graph__mode,"");
```

否则令 filename 的内容等于参数 1，并进入绘图模式。

```
format=check__extname(filename);
```

然后利用函数 check__extname 返回源图形文件的格式编号。

```
swtich(format)  {
     case 1: load__pcx(0,0,filename,0,COPY__PUT);
               break;
     case 2: load__pmt(filename,0,0,COPY__PUT);
               break;
     case 3: load__dvc(filename,0,COPY__PUT);
               break;
     case 4: load__tpc(0,0,filename,0,COPY__PUT);
               break;
     case 5: load__bpc(0,0,filename,0,COPY__PUT);
```

```
            break;
    case 6: load_map(0,0,filename,0,COPY_PUT);
            break;
    case 7: load_put(0,0,filename,0,COPY_PUT);
            break;
    case -1:closegraph();
             puts("File format error!");
             exit(1);
}
```

再利用 switch 语句针对不同的格式使用不同的图形载入格式即可。而这些图形载入格式则直接利用 3.2 节的各格式，并无需做任何的更动．剩下的储存图形的工作则交给 save_picture 函数负责。

```
    save picture();
    closegraph();
}
```

至于判别文件格式的函数 check_extname 其内容如下:

```
int check_extname(char * filename)
{
     int length;
     int i;

     length = strlen(filename);
     for(i=0;i<length;i++)
         if( * filename++)= ='.')brak;
     if(i<length) {
         for(i=0;filename[i]!='\0'; i++)
             filename[i] = toupper(filename[i]);
         if(!strcmp(filename,"PCX") )      return 1;
         else if(!strcmp(filename,"PMT") )  return 2;
         else if(!strcmp(filename,"DVC") )  return 3;
         else if(!strcmp(filename,"TPC") )  return 4;
         else if(!strcmp(filename,"BPC") )  return 5;
         else if(!strcmp(filename,"MAP") )  return 6;
         else if(!strcmp(filename,"PUT") )  return 7;
     }
}
```

这个函数的原理为:

1. 先将filename的指针指向'·'（也就是主文件名和扩展名的区分符号）的下一个字符。
2. 再将以后至字串结束符号前的所有字符全部换成大写，然后用strcmp逐一比较。

比较符合立即返回其值，并即结束本函数。如果都不符合，则返回-1。当然你也可以使用 stricmp 替代 strcmp 省略掉换成大写的动作。

3.3.2 标记图形区块

显示图形的问题解决后，接下来便是指定图形储存的区域。为了使程序便于阅读，在程序的前端定义了如下的常数：

```
#define ESC      0x011b
#define LEFT     0x4b00
#define RIGHT    0x4d00
#define UP       0x4800
#define DOWN     0x5000
#define SPACE    0x3920
#define F1       0x3b00
#define F2       0x3c00
#defien F3       0d00
#define F4       0x3e00
#define PLUS     0x4e2b
#define MINUS    0x4a2b
#define MaxX     719
#define MaxY     347
```

接着就是 save__picture 的内容：

```
void save__picture(viod)
{
    int lux,
        luy,
        rdx,
        rdy;
    int dist=1;
    int quit=0;
    int flag=1;
    int key;
    int oldlux,oldluy,oldrdx,oldrdy;
```

首先是区域变量的定义：

lux，luy	是指定区域的左上角坐标。
rdx，rdy	是右下角坐标。
dist	记载每次按箭头键所移动的点数。
quit	为判断是否离开指定区域回路的指针，如果quit=1，则跳离该循环。
flag	此变量指示按箭头键改变的是图形的大小还是图形位置，如果flag=0改变的是大小，预设值为 1。
key	记载按键的值。

oldlux，oldluy，oldrdx，oldrdy 记载上一次方框的坐标，以便消除旧的方框。

```
lux=luy=0;
rdx=MaxX;
rdy=MaxX;
detwritemode(XOR_PUT);
rectangle(lux,luy,rdx,rdy);
```

程序一开始时，先初设左上角坐标(lux,luy)为(0,0)，而右下角坐标(rdx,rdy)的值为最大值(max X,max Y)也即(719,347)。同时设定线条的写入格式为 XOR_PUT，以免在画方框时破坏了背景。因为使用 XOR_PUT 来画框时，只要在同一个地方画两遍就会还原成原来的样子，这是 XOR 的特性。

```
while(!quit){
      oldlux=lux;
      oldluy=luy;
      oldrdx=rdx;
      oldrdy=rdy;
      key=bioskey(0);
```

接着进入循环。先利用(oldlux,oldluy)及(oldrdx,oldrdy)来保存目前的坐标，以方便将来的还原动作，然后利用 bioskey(0)读取按键。

```
switch(key)  {
      case UP  :
                  if(!flag)  {
                      rdy-=dist;
                      luy-=dist;
                      if( (rdy<0)||(luy<0) )  {
                      rdy  =  oldrdy;
                      luy  =  oldluy;
                      }
                  }
                  else     {
                              rdy-=dist;
                              if(rdy<0) rdy  =  oldrdy;
                  }
                  rectangle(oldlux,oldluy,oldrdx,oldrdy);
                  rectangle(lux,luy,rdx,rdy);
                  break;
```

利用 switch 判断按了什么按键，如果是按了 UP 键，就判断 flag，看是要改变方框的大小还是位置。如果 flag=0，表示是要改变位置，那么 luy 及 rdy 都减掉 dist；如果 flag=1，表示是要改变方框的大小，那么仅需将 rdy 减掉 dist 就可以了。不管改变位置还是大小，改变完毕后都必须检查有没有超过边界 0，如果超过就要恢复原来的值，不做改变。

改变完毕，先以 rectangele(oldlux,oldluy,oldrdx,oldrdy)消去旧的方框，再用 rectangele(lux,luy,rdx,rdy)画上新的方框。

接下来的 DOWN，LEFT 及 RIGHT 都是相同的原理，如下:

```
case DOWN :
            if(!flag) {
               rdy+=dist;
               luy+=dist;
               if( (rdy>MaxX)||(luy>MaxY) ) {
               rdy = oldrdy;
               rdy = oldluy;
               }
          }
          else {
               rdy+=dist;
               if(rdy>MaxY) rdy = oldrdy;
               }
               rectangle(oldlux,oldluy,oldrdx,oldrdy);
               rectangle(lux,luy,rdx,rdy);
               break;
case LEFT :
          if(!flag) {
               rdx-=dist;
               lux-=dist;
               if( (rdx<0)||(lux<0) ){
                 rdx = oldrdx;
                 lux = oldlux;
               }
          }
          else {
               rdx-=dist;
               if(rdx<0) rdx = oldrdx;
               }
               rectangle(oldlux,oldluy,oldrdx,oldrdy);
               rectangle(lux,luy,rdx,rdy);
               break;
case RIGHT :
            if(!flag) {
                rdx+=dist;
                lux+=dist;
                if( (rdx>MaxX)||(lux>MaxX) ) {
                  rdx = oldrdx;
                  lux = oldlux;
                }
```

```
            }
            else {
                rdx+=dist;
                if(rdx>MaxX) rdx = oldrdx;
            }
            rectangle(oldlux,oldluy,oldrdx,oldrdy);
            rectangle(lux,luy,rdx,rdy);
            break;
case SPACE :
            flag = (flag) ? 0 : 1;
          break;
```

空格键(SPACE)用来切换 flag 的值。

```
case PLUS :
          dist++;
          dist=(dist>50) ? 50 : dist;
          break;
case NIMUS :
          dist--;
          dist=(dist<1) ? 1 : dist;
          break;
```

每按一次+键就将 dist 的值加 1，但将它限制在 50 以下。每按一次−键就将 dist 的值减 1，但增量 dist 不得小于 4，否则便失去意义。

```
case ESC :
          quit=1;
          break;
```

如果按下 ESC 键便令 quit=1，立即离开本循环，即结束本程序。

3.3.3 选择图形文件的储存格式

```
case F1 :
        rectangle(lux,luy,rdx,rdy);
        write_tpc(lux,luy,rdx,rdy);
        rectangle(lux,luy,rdx,rdy);
        break;

case F2 :
        rectangle(lux,luy,rdx,rdy);
        write_tpc(lux,luy,rdx,rdy);
        rectangle(lux,luy,rdx,rdy);
        break;

case F3 :
```

```
            rectangle(lux,luy,rdx,rdy);
            write__tpc(lux,luy,rdx,rdy);
            rectangle(lux,luy,rdx,rdy);
            break;

    case F4 :
            rectangle(lux,luy,rdx,rdy);
            write__tpc(lux,luy,rdx,rdy);
            rectangle(lux,luy,rdx,rdy);
            break;
            }
        }
    }
```

按下 F1～F4 任一键，就分别调用 write__tpc,write__bpc,write__map 及 write__put 将(lux,luy)-(rdx,rdy)区域以该格式储存起来。在储存之前，为了避免连同方框本身一并储存，必须先暂时消去方框（利用 rectangle(lux,luy,rdx,rdy)），等储存完毕再画上。

至于图形文件写入的 4 个函数的原理可用 write__tpe 为例说明如下

```
void write__tpc(int lux,int luy,int rdx,int rdy)
{
      int temp;
      unsigned size;
      char * buf;
      FILE * fp;
      char newfilename[13];
      int i;
```

先定义一些需要的区域变量．本函数所需要传进来的参数是储存的图形区域的左上角及右下角坐标，即(lux,luy)及(rdx,rdy)。

```
strcpy(newfilename,filename);
for(i=0; ( (newfilename[i]!='.') &&
           (newfilename[i]!='\0') ) ;i++);
newfilename[i]='\0';
strcat(newfilename,".TPC");
```

这一段程序码将原来的文件名的扩展名改变为 TPC，存在 newfilename 中

```
if(lux>rdx) { temp=lux; lux=rdx; rdx=temp; }
if(luy>rdy) { temp=luy; luy=rdy; rdy=temp; }
```

这段程序码用来确定 (lux, luy) 为左上角坐标，而(rdx,rdy)为右下角坐标 (rdx, lux, rdy, luy)．

```
size  = imageesize(lux, luy, rdx, rdy);
buf   = (char * ) mallco(size);
```

```
getimage(lux, luy, rdx, rdy, buf);
```

接着计算所需的内存大小，然后配置给 buf 指针，并利用 getimage 函数取得屏幕上的该块图形数据。

```
if( (fp=fopen(newfilename, "wb") )= =NULL) {
      closegraph();
      puts("File writting error!");
      exit(1);2
      }
fwrite(buf,size-2,1,fp);
fcolse(fp);
free(buf);
}
```

最后再打开文件、写入数据、关闭文件、释放内存，则本函数即告完成。因为 TPC 文件格式和 Turbo C 中 getimage 处理图形方式几乎一致，所以在将图形数据写入磁盘之前，并无需对 buf 所指向的内存区块做任何处理。在此唯一比较特别的地方是写入的长度为 size-2，即忽略掉 Turbo C 在 getimage 处理下所保留的 2 个字节，TPC 格式并不理会这 2 个字节，所以无需一并写入文件中。

下一个图形文件写入函数为 write__bpc。

```
void write__bpc(int lux, int luy, int rdx, int rdy)
{
      int temp;
      unsigned size;
      char  * buf;
      FILE  * fp;
      char newfilename[13];
      char mark[]={0xfd, 0x00, 0x00, 0x00, 0x00};
      char total[2];
      int i;
```

请参考 3.2.5 节，BPC 的文件格式较 TPC 多了前面七个字节，前 5 个字节恒为 FD，00，00，00，00，而剩下的两个字节则记录图形数据的长度。所以在定义变数时，较前面的 write__tpc 多定义了 mark[ ]以存放 FD，00，00，00，00 这 5 个字节，而 total 则存放经计算后转换成字节的图形数据长度。

```
strpy(newfilename, filename);
for(i=0; ( (newfilename[i]!='.')&&(newfilename[i]!='\0') );i++);
newfilename[i]='\0';
strcat(newfilename,".BPC");
if(lux>rdx) { temp=lux; lux=rdx; rdx=temp; }
if(luy>rdy) { temp=luy; luy=rdy; rdy=tmep; }
size =imagesize(lux, luy, rdx, rdy);
```

```
buf =(char *) malloc(size);
getimage(lux, luy, rdx, rdy, buf);
if(fp=fopen(newfilename, "wb") )= =NULL) {
      closegraph();
      puts("File writting error!");
      exit(1);
}
```

这一段和 write_tpc 一样。

```
fwrite(mark, 5, 1, fp);
total[0] =(char) (((size-2)%256) & 0xff);
total[1] =(char) (((size-2)/256) & 0xff);
fwrite(total, 2, 1, fp);
```

这一段先写入前 5 个字节，接着计算该图形数据的全长，total[0]存放数据长度的低字节，而 total[1]存放数据长度的高字节，再依序写入磁盘。

```
if(((unsigned) buf[0]& 0xff) !=0xff) buf[0]++;
else {
      buf[0]=0x00;
      buf[1]++;
}

if(((unsigned) buf[2] & 0xff !=0xff) buf[2]++;
else {
      buf[2]=0x00;
      buf[3]++;
}
```

另外，在写入数据时还有一点必须注意，就是在 Turbo C 中图形数据的最前端的第 0 及第 1 字节记录此图形的宽度减 1，第 2 及第 3 字节记录此图形的高度减 1，而 BPC 格式记录的是宽及高，所以在写入之前必须将这两个值分别加 1 才行。

但是由于宽高分别由两个字节表示，我们并不能单纯地把低字节（即 buf[0]及 buf[2]）各加上 1，还必须考虑到进位的情形。我们的方法是先判断低字节的值是否为 0xFF，如果是的话就将低字节归零并将高字节加 1，否则就直接将低字节加 1 就好了。用以上的步骤便可以安全地将宽及高各加上 1，而不必担心溢出而被截掉的情形。

```
    fwrite(buf,size-2,1,fp);
    fclose(fp);
    free(buf);
}
```

剩下的操作就是写入图形数据、关闭文件、释放内存了。

下一个函数 write_map 的方法和 write_bpc 几乎一模一样，唯一不同的地方是 MAP 之前 5 个字节是 FD，A1，10，00，00（指单色而言），而非 BPC 的 FD，00，

00，00，00，请看程序列表：

```
void write_map(int lux, int luy, int rdx, int rdy)
{
        int temp;
        unsigned size;
        char  * buf;
        FILE  * fp;
        char newfilename[13];
        char mark[]={0xfd, 0xa1, 0x10, 0x10, 0x00, 0x00};
        char total[2];
        int i;

        strcpy(newfilename, filename);
        for( i=0; ( (newfilename[i]!='.') &&
                    (newfilename[i]!='0') ) ; i++);
        newfilename[i]='\0';
        strcat(newfilename, ".MAP");
        if(lux>rdx) {
              temp=lux;
              lux=rdx;
              rdx=temp;
        }
        if(luy>rdy) {
              temp=luy;
              luy=rdy;
              rdy=temp;
        }
        size=imagesize(lux, luy, rdx, rdy);
        buf=(char  * ) malloc(size);
        getimage(lux, luy, rdx, rdy, buf ;

        if((fp=fopen(newfilename,"wb"))= =NULL) {
              closegraph();
              puts("File writting error!");
              exit(1);
        }
        fwrite(mark, 5, 1, fp);
        total[0] = (char) (((size-2) %256) & 0xff);
        total[1] = (char) (((size-2) /256) & 0xff);
        fwrite(total, 2, 1, fp);

        if(((unsigned) buf[0] & 0xff) != 0xff)
              buf[0]++;
```

```
    else {
        buf[0]=0x00;
        buf[1]++;
    }
    if(((unsigned) buf[2] & 0xff) !=0xff)
        buf[2]++;
    else {
        buf[2]=0x00;
        buf[3]++;
    }
    fwrite(buf, size-2, 1, fp);
    fclose(fp);
    free(buf);
}
```

最后一个函数 write_put 则和 write_tpc 相类似，除了宽高各加 1 外，其余一模一样，请看程序列表：

```
void write_put(int lux, int luy, int rdx, int rdy)
{
    int temp;
    unsigned size;
    char *buf;
    FILE *fp;
    char newfilename[13];
    int i;

    strcpy(newfilename, filename);
    for( i=0; ((newfilename[i]!='.') &&
        (newfilename[i]!='\0')) ;i++);
    newfilename[i]='\0';
    strcat(newfilename, ".PUT");
    if(lux>rdx) {
        temp=lux;
        lux=rdx;
        rdx=temp;
    }
    if(luy>rdy) {
        temp=luy;
        luy=rdy;
        rdy=temp;
    }
    size=imagesize(lux,luy,rdx,rdy,);
    buf=(char *) malloc(size);
```

```
    getimage(lux,luy,rdx,rdy,buf);

    if((fp=fopen(newfilename, "wb"))==NULL) {
        closegraph();
        puts("File writtine error!");
        exit(1);
    }
    if(((unsigned) buf[0] & 0xff) !=0xff)
        buf[0]++;
    else {
        buf[0]=0x00;
        buf[1]++;
    }
    if(((unsigned) buf[2] & 0xff) !=0xff)
        buf[2]++;
    else  {
        buf[2]=0x00;
        buf[3]++;
    }
    fwrite(buf,size-2,1,fp);
    fclose(fp);
    free(buf);
}
```

本节的最后附上全部源程序 GTRANS.C，请参考程序列表：

```
/* GTRANS.C */

#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <bios.h>
#include <alloc.h>

#define ESC      0x011b
#define LEFT     0x4b00
#define RIGHT    0x4d00
#define UP       0x4800
#define DOWN     0x5000
#define SPACE    0x3920
#define F1       0x3b00
#define F2       0x3c00
#define F3       0x3d00
#define F4       0x3e00
```

```
#define PLUS        0x4e2b
#define MINUS       0x4a2d
#define MaxX        719
#define MaxY        347

void load_pcx(int x, int y, char *filename, int page, int status);
void load_pmt(char *filename, int page, int status);
void load_dvc(char *filename, int page, int status);
void load_tpc(int x, int y, char *filename, int page, int status);
void load_bpc(int x, int y, char *filename, int page, int status);
void load_map(int x, int y, char *filename, int page, int status);
void load_put(int x, int y, char *filename, int page, int status);
void write_tpc(int lux, int luy, int rdx, int rdy);
void write_bpc(int lux, int luy, int rdx, int rdy);
void write_map(int lux, int luy, int rdx, int rdy);
void write_put(int lux, int luy, int rdx, int rdy);
int  check_extname(char *filename);
void save_picture(void);

char filename[13];

main(argc, argv)
int argc;
char *argv[];
{
    int graph_driver=HERCMONO, graph_mode=HERCMONOHI;
    int format;

    if(argc<2) {
        puts("Usage : GTRANS [graphics filename]")
        exit(1);
    }
    strcpy(filename, argv[1]);
    initgraph(&graph_driver, &graph_mode, "");
    format = check_extname(filename);
    switch(format) {
        case 1 : load_pcx(0,0,filename,0,COPY_PUT);
                 break;
        case 2 : load_pmt(filename,0,COPY_PUT);
                 break;
        case 3 : load_dvc(filename,0,COPY_PUT);
                 break;
        case 4 : load_tpc(0,0,filename,0,COPY_PUT);
                 break;
```

```
            case 5 : load__bpc(0,0,filename,0,COPY__PUT);
                     break;
            case 6 : load__map(0,0,filename,0,COPY__PUT);
                     break;
            case 7 : load__put(0,0,filename,0,COPY__PUT);
                     break;
            case -1: closegraph();
                     puts("File format error!");
                     exit(1);
            }
            save__picture();
            closegraph();
}

int check__extname(char *filename)
{
        int  length;
        int  i;

        length = strlen(filename);
        for(i=0; i<length; i++)
            if(*(filename++)=='.') break;
        if(i<length) {
            for(i=0; filename[i]!='\0'; i++)
                filename[i] = toupper(filename[i]);
            if(!strcmp(filename,"PCX"))         return 1;
            else if(!strcmp(filename, "PMT")) return 2;
            else if(!strcmp(filename, "DVC")) return 3;
            else if(!strcmp(filename, "TPC")) return 4;
            else if(!strcmp(filename, "BPC")) return 5;
            else if(!strcmp(filename, "MAP")) return 6;
            else if(!strcmp(filename, "PUT")) return 7:
            else return -1;
            }
}

void save__picture(void)
{
        int lux,        /* left-up coordinate X */
            luy,        /* left-up coordinate Y */
            rdx,        /* right-down coordinate X */
            rdy;        /* right-down coordinate Y */
        int dist=1;     /* distance */
        int quit=0;
```

```
    int flag=1;
    int key;
    int oldlux, oldluy, oldrdx, oldrdy;

    lux = luy = 0;
    rdx = MaxX;
    rdy = MaxY;
    setwritemode(XOR_PUT);
    rectangle(lux, luy, rdx, rdy);
    while(!quit) {
        oldlux = lux;
        oldluy = luy;
        oldrdx = rdx;
        oldrdy = rdy;
        key = bioskey(0);
        switch(key) {
            case UP   : /* move to up */
                        if(!flag) {
                            rdy-=dist;
                            luy-=dist;
                            if((rdy<0)||(luy<0)) {
                                rdy = oldrdy;
                                luy = oldluy;
                            }
                        }
                        else   {
                                rdy-=dist;
                                if(rdy<0) rdy = oldrdy;
                        }
                        rectangle(oldlux, oldluy, oldrdx, oldrdy);
                        rectangle(lux, luy, rdx, rdy);
                        break;

            case DOWN : /* move to down */
                        if(!flag) {
                            rdy+=dist;
                            luy+=dist;
                            if((rdy>MaxY)||(luy>MaxY)) {
                                rdy = oldrdy;
                                luy = oldluy;
                            }
                        }
                        else {
                            rdy+=dist;
```

```
                    if(rdy>MaxY) rdy = oldrdy;
                }
                rectangle(oldlux, oldluy, oldrdx, oldrdy);
                rectangle(lux, luy, rdx, rdy);
                break;

        case LEFT : / * move to left * /
                if(!flag) {
                rdx-=dist;
                lux-=dist;
                if((rdx<0)||(lux<0)) {
                        rdx = oldrdx;
                        lux = oldlux;
                }
                }
                else {
                        rdx-=dist;
                        if(rdx<0) rdx = oldrdx;
                }
                rectangle(oldlux, oldluy, oldrdx, oldrdy);
                rectangle(lux, luy, rdx, rdy);
                break;

        case RIGHT: / * move to right * /
                if(!flag) {
                        rdx+=dist;
                        lux+=dist;
                        if((rdx>MaxX)||(lux>MaxX)) {
                rdx = oldrdx;
                        lux = oldlux;
                        }
                }
                else {
                        rdx+=dist;
                        if(rdx>MaxX) rdx = oldrdx;
                }
                rectangle(oldlux, oldluy, oldrdx, oldrdy);
                rectangle(lux, luy, rdx, rdy);
                break;

        case SPACE: / * toggle the coordinate to move * /
                flag = (flag) ? 0 : 1;
                break;

```

```
            case PLUS : / * plus the distance * /
                    dist ++;
                    dist = (dist>50) ? 50 : dist;
                    break;

            case MINUS: / * minus the distance * /
                    dist--;
                    dist = (dist<1) ? 1 : dist;
                    break;

            case ESC  : / * quit the program * /
                    quit = 1;
                    break;

            case F1   : / * save the picture to TPC format * /
                    rectangle(lux, luy, rdx, rdy);
                    write_tpc(lux, luy, rdx, rdy);
                    rectangle(lux, luy, rdx, rdy);
                    break;

            case F2   : / * save the picture to BPC format * /
                    rectangle(lux, luy, rdx, rdy);
                    write_bpc(lux, luy, rdx, rdy);
                    rectangle(lux, luy, rdx, rdy);
                    break;

            case F3   : / * save the picture to MAP format * /
                    rectangle(lux, luy, rdx, rdy);
                    write_map(lux, luy, rdx, rdy);
                    rectangle(lux, luy, rdx, rdy);
                    break;

            case F4   : / * save the picture to PUT format * /
                    rectangle(lux, luy, rdx, rdy);
                    write_put(lux, luy, rdx, rdy);
                    rectangle(lux, luy, rdx, rdy);
                    break;
            }
      }
}

void load_pcx(int x, int y, char *filename, int page, int status)
{
      typedef struct {
```

```
            char manufacturer;
            char version;
            char encoding;
            char bits_per_pixel;
            int  xmin, ymin;
            int  xmax, ymax;
            int  hres;
            int  vres;
            char palette[48];
            char reserved;
            char nplanes;
            int bits_per_line;
            int  palette_type;
            char filler[58];
                } PCX_HEAD;
    FILE  * fp;
    int i;
    char picdata;
    unsigned xlength, ylength, xlength_b, size, cnt=0, total=0;
    PCX HEAD header;
    char  * buf;
    int  * s;

    if((fp = fopen(filename, "rb"))==NULL) {
          closegraph();
          puts("File Reading Error!!");
          exit(1);
    }
    fread((char  * ) &header, 1, sizeof(PCX_HEAD), fp);
    xlength = header.xmax - header.xmin + 1;
    ylength = header.ymax - header.ymin + 1;
    if((xlength>720)||(ylength>348)) {
    closegraph();
    puts("Picture is too large!");
    exit(1);
    }
    size = imagesize( header.xmin, header.ymin,
                      header.xmax, header.ymax);
    buf = (char  * ) malloc(size);
    s = (int  * ) buf;
    buf += 4;          / *  Reserve the space for length of x and y  * /
    if(xlength%8)
    xlength b = (xlength >> 3) + 1;
    else
```

```
        xlength_b = xlength >> 3;

        while(total < xlength_b * ylength) {
        cnt = 1;
        picdata = fgetc(fp);
        if(0xc0 = = (0xc0 & picdata)) {
                cnt = 0x3f & picdata;
                picdata = fgetc(fp);
        }
        for(i = 0; i < cnt; i++)
                *(buf++) = picdata;
        total += cnt;
        }
        fclose(fp);
        setactivepage(page);
        s[0] = xlength - 1;
        s[1] = ylength - 1;
        putimage(x, y, s, status);
        free(s);
}

void load_pmt(char *filename, int page, int status)
{
FILE *fp;
unsigned int counter, number, cnt, x, y;
register int i;
char data;
char *buf;
unsigned size;

size = imagesize(0,0,getmaxx(), getmaxy());
buf = (char *) malloc(size);
if(buf = = NULL) {
closegraph();
puts("Memory is not enough");
exit(1);
}
buf[0] = 719 % 256;             /* The width of picture */
buf[1] = 719 / 256;
buf[2] = 347 % 256;             /* The height of picture */
buf[3] = 347 / 256;
if((fp = fopen(filename, "rb")) = = NULL) {
closegraph();
puts("File reading error!");
```

```
exit(1);
}
fread(&number, sizeof(int), 1, fp);
counter = 0;
y = 0;
while(1)
{
x = 0;
while(x<216)
{
    data = fgetc(fp);
    counter ++;
    cnt = data & 0x7f;
    if(!(data>>7))
    {                /* if MSB of data is clear */
    for(i=0; i<cnt; i++)
    {
    if(x<90)
        buf[y*90+x+4] = fgetc(fp) & 0xff;
    else
        fgetc(fp);
    x ++;
    if(x>=216)
    {
        x = 0;
        y ++;
        if(y==348) goto end;
    }
    }
    counter += cnt;
    }
    else
    {                /* if MSB of data is set */
    data = fgetc(fp);
    counter ++;
    for(i=0; i<cnt; i++)
     {
    if(x<90)
        buf[y*90+x+4] = data & 0xff;
    x ++;
    if(x>=216)
    {
        x = 0;
        y ++;
```

```
            if(y= =348) goto end;
      }
      }
      }
      if(counter> =number)
      {
      fread(&number, sizeof(int), 1, fp);
      counter = 0;
      }
}
}
end:;
setactivepage(page);
putimage(0,0,buf,status);
free(buf);
}

void load_dvc(char *filename, int page, int status)
{
      FILE *fp;
      register int i;
      int str_length, counter_number, index;
      int counter_pointer[60], pointer=1;
      int str_pointer=0;
      int rept;
      char data;
      int x=0, y=0;
      char string[50];
      char *buf;
      unsigned size;

      size = imagesize(0,0,getmaxx(), getmaxy());
      buf = (char *) malloc(size);  /* allocate memory of one page */
      if(buf= =NULL) {
            closegraph();
            puts("Memory is not enough");
            exit(1);
      }

      buf[0] = 639 % 256;         /* The width of picture */
      buf[1] = 639 / 256;
      buf[2] = 347 % 256;         /* The height of picture */
      buf[3] = 347 / 256;

```

```
if((fp = fopen(filename, "rb"))==NULL)
{
        closegraph();
        puts("\aFile reading error!");
        exit(0);
}

for(i=0; i<12; i++)                    /* skip the file header */
        fgetc(fp);
for(y=0; y<348; y++) {
x = 0;
index = 0;
pointer = 1;
str_pointer = 0;
str_length      = (int) fgetc(fp) & 0xff;
counter_number = (int) fgetc(fp) & 0xff;
for(i=0; i<counter_number; i++)
        counter_pointer[i] = (int) fgetc(fp) & 0xff;
counter_pointer[i] = -1;
str_pointer = counter_number + 1;
while(str_pointer<str_length)            {
        if(counter_pointer[index]==pointer)  {
                index ++;
                rept = (int) fgetc(fp) & 0xff;
                rept = (rept>80) ? 80 : rept;
                data = fgetc(fp);
                str_pointer += 2;
                pointer += 2;
        }
        else                {
                rept = 1;
                data = fgetc(fp);
                str_pointer ++;
                pointer ++;
        }
        for(i=x; i<x+rept; i++)
                buf[y*80+i+4] = data;
        x += rept;
}
}
fclose(fp);
setactivepage(page);
putimage(0,0,buf,status);
free(buf);
```

```
}

void load_tpc(int x, int y, char *filename, int page, int status)
{
    int *buf;
    unsigned size;
    FILE *fp;
    int i;

    if((fp = fopen(filename, "rb"))==NULL) {
        closegraph();
        puts("File reading error!");
        exit(1);
    }

    size = filelength(fileno(fp));
    buf = (int *) malloc(size);
    if(buf==NULL) {
        closegraph();
        puts("Memory is not enough!");
        exit(1);
    }
    fread(buf, 1, size, fp);
    fclose(fp);
    setactivepage(page);
    putimage(x,y,buf,status);
    free(buf);
}

void load_bpc(int x, int y, char *filename, int page, int status)
{
    int *buf;
    unsigned size;
    FILE *fp;
    int i;

    if((fp = fopen(filename, "rb"))==NULL) {
    closegraph();
    puts("File reading error!");
    exit(1);
    }
    for(i=0; i<7; i++)
    fgetc(fp);
    size = filelength(fileno(fp));
```

```
        buf = (int *) malloc(size);
        if(buf==NULL) {
        closegraph();
        puts("Memory is not enough!");
        exit(1);
        }
        fread(buf, 1, size, fp);
        fclose(fp);
        buf[0]--;
        buf[1]--;
        setactivepage(page);
        putimage(x,y,buf,status);
        free(buf);
}

void load_map(int x, int y, char *filename, int page, int status)
{
        int *buf;
        unsigned size;
        FILE *fp;
        int i;

        if((fp = fopen(filename, "rb"))==NULL) {
        closegraph();
        puts("File reading error!");
            exit(1);
        }
        for(i=0; i<7; i++)
        fgetc(fp);
        size = filelength(fileno(fp));
        buf = (int *) malloc(size);
        if(buf==NULL) {
        closegraph();
        puts("Memory is not enough!");
        exit(1);
        }
        fread(buf, 1, size, fp);
        fclose(fp);
        buf[0]--;
        buf[1]--;
        setactivepage(page);
        putimage(x,y,buf,status);
        free(buf);
}
```

```
void load_put(int x, int y, char *filename, int page, int status)
{
       int *buf;
       unsigned size;
       FILE *fp;
       int i;

       if((fp = fopen(filename, "rb"))==NULL) {
       closegraph();
       puts("File reading error!");
       exit(1);
       }
       size = filelength(fileno(fp));
       buf = (int *) malloc(size);
       if(buf==NULL) {
       closegraph();
       puts("Memory is not enough!");
       exit(1);
       }
       fread(buf, 1, size, fp);
       fclose(fp);
       buf[0]--;
       buf[1]--;
       setactivepage(page);
       putimage(x,y,buf,status);
       free(buf);
}

void write_tpc(int lux, int luy, int rdx, int rdy)
{
       int temp;
       unsigned size;
       char *buf;
       FILE *fp;
       char newfilename[13];
       int i;

       strcpy(newfilename, filename);
       for( i=0; ( (newfilename[i]!='.') &&
                   (newfilename[i]!='\0') ) ; i++ );
       newfilename[i]='\0';
       strcat(newfilename, ".TPC");    /* change the ext. filename */
       if(lux>rdx) { temp=lux; lux=rdx; rdx=temp; }
```

```
        if(luy>rdy) { temp=luy; luy=rdy; rdy=temp; }
        size = imagesize(lux, luy, rdx, rdy);
        buf = (char * ) malloc(size);          / * allocate memory * /
        getimage(lux, luy, rdx, rdy, buf); / * get the picture into buffer * /
        if((fp=fopen(newfilename, "wb"))= =NULL) {
        closegraph();
        puts("File writing error!");
        exit(1);
        }
        fwrite(buf, size-2, 1, fp);          / * cast away the last 2 byte * /
        fclose(fp);                            / * that Turbo C reserved * /
        free(buf);
}

void write_bpc(int lux, int luy, int rdx, int rdy)
{
int temp;
unsigned size;
char * buf;
FILE * fp;
char newfilename[13];
char mark[]={0xfd, 0x00, 0x00, 0x00, 0x00};
char total[2];
int i;

strcpy(newfilename, filename);
for(i=0; ((newfilename[i]!='.')&&(newfilename[i]!='\0')) ; i++) ;
newfilename[i]='\0';
strcat(newfilename, ".BPC");          / * change the ext. filename * /
if(lux>rdx) { temp=lux; lux=rdx; rdx=temp; }
if(luy>rdy) { temp=luy; luy=rdy; rdy=temp; }
size = imagesize(lux, luy, rdx, rdy);
buf = (char * ) malloc(size);          / * allocate memory * /
getimage(lux, luy, rdx, rdy, buf);  / * get the picture into buffer * /
if((fp=fopen(newfilename, "wb"))= =NULL) {
        closegraph();
        puts("File writing error!");
        exit(1);
}
fwrite(mark, 5, 1, fp);          / * write the mark FD 00 00 00 00 * /
total[0] = (char) (((size-2) % 256) & 0xff);
total[1] = (char) (((size-2) / 256) & 0xff);

fwrite(total, 2, 1, fp);          / * write the total file size * /
```

```
if(((unsigned) buf[0] & 0xff) != 0xff) buf[0]++;
else {
        buf[0]=0x00;
          buf[1]++;                         / * increment picture width * /
}

if(((unsigned) buf[2] & 0xff) != 0xff) buf[2]++;
else {
        buf[2]=0x00;                      / * increment picture height * /
        buf[3]++;
}
fwrite(buf, size-2, 1, fp);          / * cast away the last 2 byte * /
fclose(fp);                            / * that Turbo C reserved * /
free(buf);
}

void write_map(int lux, int luy, int rdx, int rdy)
{
        int temp;
        unsigned size;
        char * buf;
        FILE * fp;
        char newfilename[13];
        char mark[]={0xfd, 0xa1, 0x10, 0x00, 0x00};
                                          / * .MAP mono picture mark * /
        char total[2];
        int i;

        strcpy(newfilename, filename);
        for( i=0; ((newfilename[i]!='.') &&
                (newfilename[i]!='\0')) ; i++) ;
        newfilename[i]='\0';
        strcat(newfilename, ".MAP");  / * change the ext. filename * /
        if(lux>rdx) {
                temp=lux;
                lux=rdx;
                rdx=temp;
        }
        if(luy>rdy) {
                temp=luy;
                luy=rdy;
                rdy=temp;
        }

```

```
        size = imagesize(lux, luy, rdx, rdy);
        buf = (char * ) malloc(size);          / * allocate memory * /
        getimage(lux, luy, rdx, rdy, buf);
                                      / * get the picture into buffer * /
        if((fp=fopen(newfilename, "wb"))==NULL) {
                closegraph();
                puts("File writing error!");
                exit(1);
        }
        fwrite(mark, 5, 1, fp);   / * write the mark FD A1 10 00 00 * /
        total[0] = (char) (((size-2) % 256) & 0xff);
        total[1] = (char) (((size-2) / 256) & 0xff);
        fwrite(total, 2, 1, fp);        / * write the total file size * /

        if( ((unsigned) buf[0] & 0xff) != 0xff )
                buf[0]++;
        else {
                buf[0]=0x00;
                buf[1]++;                   / * increment picture width * /
        }
        if(((unsigned) buf[2] & 0xff) != 0xff)
                buf[2]++;
        else {
                buf[2]=0x00;                / * increment picture height * /
                buf[3]++;
        }
        fwrite(buf, size-2, 1, fp);       / * cast away the last 2 byte * /
        fclose(fp);                        / * that Turbo C reserved * /
        free(buf);
}

void write_put(int lux, int luy, int rdx, int rdy)
{
        int temp;
        unsigned size;
        char * buf;
        FILE * fp;
        char newfilename[13];
        int i;

        strcpy(newfilename, filename);
        for( i=0; ((newfilename[i]!='.') &&
        (newfilename[i]!='\0')) ; i++) ;
        newfilename[i]='\0';
```

```
724:    strcat(newfilename, ".PUT");        /* change the ext. filename */
725:    if(lux>rdx) {
726:            temp=lux;
727:            lux=rdx;
728:            rdx=temp;
729:    }
730:
731:    if(luy>rdy) {
732:            temp=luy;
733:            luy=rdy;
734:            rdy=temp;
735:    }
736:    size = imagesize(lux, luy, rdx, rdy);
737:    buf = (char *) malloc(size);           /* allocate memory */
738:    getimage(lux, luy, rdx, rdy, buf);
739:                                 /* get the picture into buffer */
740:
741:    if((fp=fopen(newtuename, "wb"))==NULL) {
742:    closegraph();
743:            puts("File writing error!");
744:            exit(1);
745:    }
746:    if(((unsigned) buf[0] & 0xff) != 0xff)
747:            buf[0]++;
748:    else {
749:            buf[0]=0x00;
750:            buf[1]++;                  /* increment picture width */
751:    }
752:    if(((unsigned) buf[2] & 0xff) != 0xff)
753:            buf[2]++;
754:    else {
755:            buf[2]=0x00;               /* increment picture height */
756:            buf[3]++;
757:    }
758:    fwrite(buf, size-2, 1, fp);        /* cast away the last 2 byte */
759:    fclose(fp);                         /* that Turbo C reserved */
760:    free(buf);
761:}
```

程序说明:

48—54 行　　取得输入的图形文件名称。

56—73 行　　判断文件格式。

78—96 行　　确认文件格式。

117–232 行　设定储存区域及文件格式.
237–303 行　装入 PCX 图形文件.
305–387 行　装入 PMT 图形文件.
389–460 行　装入 DVC 图形文件.
462–487 行　装入 TPC 图形文件.
489–517 行　装入 BPC 图形文件.
519–547 行　装入 MAP 图形文件.
549–547 行　装入 PUT 图形文件.
577–604 行　将屏幕所显示或标记的图形以 TPC 格式储存.
606–650 行　将屏幕所显示或标记的图形以 BPC 格式储存.
652–709 行　将屏幕所显示或标记的图形以 MAP 格式储存.
711–761 行　将屏幕所显示或标记的图形以 PUT 格式储存.

图 3–11　图形转换 GTRANS.C 的执行实例

3.4　图形文件的解读技巧

前文曾经提过，各种绘图软件均有自己的文件格式，在运用之前，如果不知道这种格式，可以先找找有无相关数据可供参考．一般而言，像 PUT，MAP 这种属于小图的区块图形文件，在手册上都会公布其格式以供用户参考，然而像 PMT，DVC 这种存大图的

117–232 行　设定储存区域及文件格式.
237–303 行　装入 PCX 图形文件.
305–387 行　装入 PMT 图形文件.
389–460 行　装入 DVC 图形文件.
462–487 行　装入 TPC 图形文件.
489–517 行　装入 BPC 图形文件.
519–547 行　装入 MAP 图形文件.
549–547 行　装入 PUT 图形文件.
577–604 行　将屏幕所显示或标记的图形以 TPC 格式储存.
606–650 行　将屏幕所显示或标记的图形以 BPC 格式储存.
652–709 行　将屏幕所显示或标记的图形以 MAP 格式储存.
711–761 行　将屏幕所显示或标记的图形以 PUT 格式储存.

图 3–11　图形转换 GTRANS.C 的执行实例

3.4　图形文件的解读技巧

前文曾经提过，各种绘图软件均有自己的文件格式，在运用之前，如果不知道这种格式，可以先找找有无相关数据可供参考．一般而言，像 PUT，MAP 这种属于小图的区块图形文件，在手册上都会公布其格式以供用户参考，然而像 PMT，DVC 这种存大图的

压缩图形文件，可就是各家的不传之秘，除非有书籍杂志专文介绍，否则还难以窥出其中奥秘呢！虽然如此，本节还是在此提出一些看出文件格式的相关技巧，俾使读者能有解决问题的能力，毕竟等别人破解文件格式，倒不如自己亲自动手，既可获得数据又学得技术，岂不是一举两得。

在使用这些技巧之前，首先你必须先有以下的工具：

1. 一个显示文件内容的程序，DEBUG 或 PCTOOLS 皆可。
2. 一部打印机。
3. 一台工程用计算机，可做 16-10-2 进制的转换。
4. 一个清醒的头脑。

好了，让我们开始吧！

以下就以绘图软件达文西的 DVC 文件解析过程来说明分析一种图形格式的方法。

要解析一种文件格式，当然要先取得该软件的相关信息。所以在实际观察文件内容之前，最好先查查使用手册，或是进入达文西决定版记录下它的相关参数。我们的观察结果如下：

软件名称：达文西一代决定版

出品公司：新人类

图形颜色：单色

全页宽度：640pixels(80bytes)

全页高度：500lines

接着我们在此环境下，储存几个较单纯的图形以利观察，包括整页黑色(black.dvc)，整页白色（white.dvc）以及全页网点（dot.dvc）。为什么选择这三种图形呢？因为在我们的脑中早已熟知这三种图形未压缩前的样子，即全部填 00（黑色）、FFh（白色）、以及 55h（密点），在实际看压缩过的文件内容时，我们便可以把这些内容拿来和我们印象中的样子做比较，看看压缩前和压缩后究竟有什么样的变化。

然而有了上列三个文件仍然不够，因为这三个文件全部属于高度规律的结构，压缩后的格式可能会具有相同的形式，使我们无法看出其它的特性，所以在此我们又储存另外两张结构简单但不属于前述高度规律之结构的图形，包括一张左上角为如下所示的图形(eight.dvc)，以及第一条扫描线为不连续横线的图形（line.dvc），如下所示：

现在实际观察它们的文件内容如下：

```
[Richard Ho] C:\DVC>
[Richard Ho] C:\DVC>debug white.dvc
-d
5AFE:0100  0B 48 73 75 4D 61 70 00-50 00 F4 01 04 01 01 50   .HsuMap.P......P
5AFE:0110  FF 04 01 01 50 FF 04 01-01 50 FF 04 01 01 50 FF   ...P....P....P.
5AFE:0120  04 01 01 50 FF 04 01 01-50 FF 04 01 01 50 FF 04   ..P....P....P..
5AFE:0130  01 01 50 FF 04 01 01 50-FF 04 01 01 50 FF 04 01   .P....P....P...
```

```
5AFE:0140  01  50  FF 04  01   01  50  FF-04 01  01  50  FF 04  01  01    P....P....P....
5AFE:0150  50  FF 04  01  01   50  FF 04-01  01  50  FF 04  01  01  50    P....P....P....P
5AFE:0160  FF  04  01  01  50   FF 04  01-01  50  FF  04  01  01  50  FF   ...P....P....P.
5AFE:0170  04  01  01  50  FF   04  01  01-50  FF 04  01  01  50  FF 04    ..P....P....P..
-
[Richard Ho] C:\DVC>
[Richard Ho] C:\DVC>debug black.dvc
-d
5AFE:0100  0B  48  73  75  4D  61  70  00-50  00  F4  01  04  01  01  50   .HsuMap.P......P
5AFE:0110  00  04  01  01  50   00  04  01-01  50  FF04  01  01  50  00    ....P....P....P.
5AFE:0120  04  01  01  50  00   04  01  01-50  00  04 01  01  50  00  04    ...P....P....P..
5AFE:0130  01  01  50  00  04   01  01  50-00  04  01 01  50  00  04  01    ..P....P....P...
5AFE:0140  01  50  00  04  01   01  50  00-04  01  01 50  00  04  01  01    .P....P....P....
5AFE:0150  50  00  04  01  01   50  00  04-01  01  50 00  04  01  01  50    P....P....P....P
5AFE:0160  00  04  01  01  50   00  04  01-01  50  00 04  01  01  50  00    ....P....P....P.
5AFE:0170  04  01  01  50  00   04  01  01-50  00  04 01  01  50  00  04    ...P....P....P..
-
[Richard Ho] C:\DVC>
[Richard Ho] C:\DVC>debug dot.dvc
-d
5AFE:0100  0B  48  73  75  4D  61  70  00-50  00  F4  01  04  01  01  50   .HsuMap.P......P
5AFE:0110  55  04  01  01  50   AA 04  01-01  50  AA 04  01  01  50  AA    ....P....P....P.
5AFE:0120  04  01  01  50  55   04  01  01-50  AA 04  01  01  50  55  04   ...P....P....P..
5AFE:0130  01  01  50  AA 04   01  01  50-55  04  01  01  50  AA 04  01    ..P....P....P...
5AFE:0140  01  50  55  04  01   01  50  AA-04 01  01  50  55  04  01  01    .P....P....P....
5AFE:0150  50  AA 04  01  01   50  55  04-01  01  50  AA 04  01  01  50    P....P....P....P
5AFE:0160  55  04  01  01  50   AA 04  01-01  50  55  04  01  01  50  AA   ....P....P....P.
5AFE:0170  04  01  01  50  FF   04  01  01-50  AA 04  01  01  50  55  04   ...P....P....P..
[Richard Ho] C:\DVC>debug eight.dvc
-d
5AFE:0100  0B  48  73  75  4D  61  70  00-50  00  F4  01  04  01  01  50   .HusMap.P......P
5AFE:0110  FF  0E  01  0B  80   7F  80  7F-80  7F  80  7F  C0  3F  46  FF   .............?F.
5AFE:0120  0D  01  0A  FF  80   7F  80  7F-80  7F  80  3F  47  FF  0E  01   ...........?G...
5AFE:0130  0B  80  7F  80  7F   80  7F  80-7F  C0  3F  46  FF  04  01  01   ..........?F....
5AFE:0140  50  FF 04  01  01   50  FF 04-01  01  50  FF 04  01  01  50    P....P....P....P
5AFE:0150  FF  04  01  01  50   FF 04  01-01  50  FF  04  01  01  50  FF   ....P....P....P.
5AFE:0160  04  01  01  50  FF   04  01  01-50  FF 04  01  01  50  FF 04    ...P....P....P..
5AFE:0170  01  01  50  FF 04   01  01  50-FF 04  01  01  50  FF 04  01     ..P....P....P...
-
[Richard Ho] C:\DVC>
[Richard Ho] C:\DVC>debug line.dvc
-d
5AFE:0100  0B  48  73  75  4D  61  70  00-50  00  F4  01  14  05  01  04   .HsuMap.P......
5AFE:0110  07  0A  0D 08  FF   F0  09  00-01  0D FF  80  10  00  01  1E    ...............
```

```
5AFE:0120  FF 04 01 01 50  FF 04 01-01 50  FF 04 01 01 50  FF   ....P....P....P.
5AFE:0130  04 01 01 50 FF  04 01 01-50  FF 04 01 01 50  FF 04   ...P....P....P..
5AFE:0140  01 01 01 FF 04  01 01 50-FF 04  01 01 50  FF 04 01   ..P....P....P...
5AFE:0150  01 50 FF 04 01  01 50 FF-04 01  01 50 FF 04 01 01   .P....P....P....
5AFE:0160  50 FF 04 01 01  50 FF 04-01 01  50 FF 04 01 01 50   P....P....P....P
5AFE:0170  FF 04 01 01 50  FF 04 01-01 50  FF 04 01 01 50 FF   ....P....P....P.
```

不知读者在观察了上列的五个文件之后，可看出什么端倪？由于篇幅的限制，我们无法将文件的全部内容列出来，有兴趣的读者可以自行列出，相信将会更清楚是怎么回事。不过由于五个文件格或大部分都是有规律的内容，所以比较前面的部分也就够了。

大部分的图形文件格式都会在文件最前端加上一些识别码及相关参数，压缩图形文件更是这样。比较上列五个文件我们便发现了这个现象，因为前面的 12 个字节全部一样，我们将它列出如下：

```
0b 48 73 75 4d 61 70 00-50 00 f4 01 14 05 01 04    .HsuMap.P.....
0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

1. 首先看第0个字节:0Bh，十进制值是11，往后数发现正好是文件头参数结束的位置，可见这个字节的功能便是记录文件头参数的长度。
2. 接下来的第1到第6个字节可能就是所谓的识别码了，因为它们都是有意义的英文字母 HsuMap。
3. 第 7 个字节 00h 可能是保留的字节（因为看不出它所代表的意义）。
4. 第8第9个字节50h 00h，其中50h为低字节，00h为高字节，计算其值为80d，可见代表的就是本张图的宽度 80 个字节，也就是 640pixels。紧接着的两个字节用同样的方法得 500，正好是扫描线数。到此，DVC 文件的文件头部分算是大致解出。

在此特别说明一点：以上所做的推论纯属凭经验上的猜测及实际上的印证，然而是不是一定就是完全正确呢？恐怕不见得，真正的正确的数据只有该软件公司才有吧！除非拥有该软件公司所公布的数据或技术手册，否则，多看多做验证，大概就是逼近正确的最佳法则。

接着是实际图形数据的压缩部分。在前三个文件发现了一个重要的特性，亦即不断地重复如下的字串：

white.dvc	black.dvc	dot.dvc
04 01 01 50 FF	04 01 01 50 FF	04 01 01 50 FF

其中一看便知 04 一定是紧接着的描述字串的长度，而 50h＝80，表示重复 80 次的 FFh，00h 或 55h，因为我们已经知道达文西的每条扫描线有 640 个 pixel，共占 80 个 byte，所以推测 DVC 文件的压缩方式应是以每条扫描线为一个压缩字串的单位。至于两个 01 是代表什么意思呢？让我们来看更多的例子。

先看看 eight.dvc 的例子：（画底线部分）

0E	01	0B	80	7F	80	7F	80	7F	80	7F	C0	3F	56	FF
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
			(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	

第 0 个字节的 0Eh＝14 代表描述字串的长度。

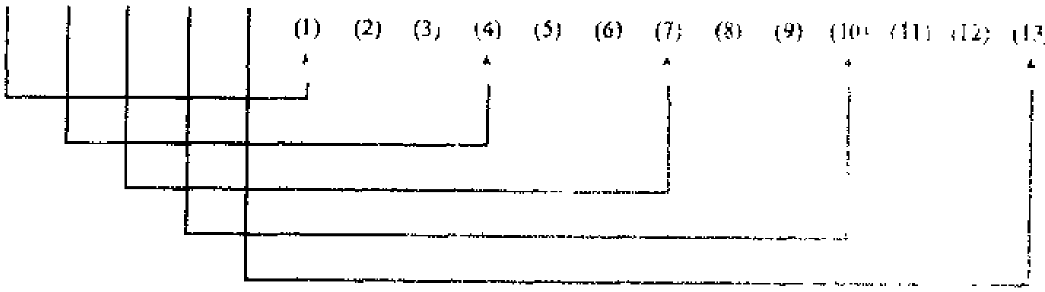
第 2 个字节 0Bh＝11，如果从 0B 后的字节开始计数的话正好指到 46h，而 46h 正是其后 FF 的重复次数。

如何得知的呢？因为 46h＝70，再加上 80 7F 80 7F 80 7F 80 7F C0 3F 等 10 个字节，正好是 80 字节，为一条扫描线的长度。而 80 7F...C0 3F 将它换成图形（即转换为二进制）如下：

80	7F	80	7F	80
↓	↓	↓	↓	↓
10000000	01111111	10000000	01111111	10000000
○●●●●●●●	●○○○○○○○	○●●●●●●●	●○○○○○○○	○●●●●●●●

是不是就是我们所绘制图形的第一条扫描线的样子呢？再来看一个例子。（line.dvc）

14	05	01	04	07	0A	0D	08	FF	F0	09	00	01	0D	FF	80	10	00	01	1E	FF
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
							(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)



这是一条扫描线中包含有两段不连续直线压缩的例子。和前几个例子不同的是：第 1 个字节不再是 01，而是 05。究竟代表着什么意义呢？各位由上面的图解是否就恍然大悟了？原来它所代表的便是这个压缩字串有几个计数器，而紧在第 1 个字节后的便是指向这些计数器的指针。被这些指针所指到的字节便是表示将重复字的下一个字节的次数，而没被指到的字节也不是计数器，下一个的字节便是重复的字节。换句话说，以本例子而言：

08　是计数器。

FF　是将被重复8次的字节。

F0　是不重复数据。

09　为计数器，00将被重复9次。

01　为不重复数据，依此类推，展开后

$$\underset{}{08}\ \underset{(1)(2)(3)}{+1}\ +09\ \underset{(4)(5)}{}\ +1\ \underset{(6)}{}\ +0D\ \underset{(7)(8)}{}\ +1\ \underset{(9)}{}\ +10\ \underset{(10)(11)}{}\ +1\ \underset{(12)}{}\ +1E\ \underset{(13)(14)}{}\ =80$$

正好等于一条扫描线的字节数.

至此，DVC 文件格式已分析完毕. 接下来的步骤便是写一个程序来印证这个分析方法是否有效. 当然，如果为了谨慎起见，可以自行找一个较为复杂的图形文件，将本套方法应用上去，核算看看是否每条扫描线都能正确地展开为 640pixel(80 byte).

要分析一种文件格式，除了必须具备几分能力及经验外，尚需要几分运气。由于压缩的方式有很多种算法可以使用，如果用这种 Run-Length 的角度还看不出什么端倪的话，也许是使用别的方法（如 LZW 或 Huffman Method）压缩的. 这些方法就留待下次有机会再为各位读者分析了.

第四章　彩色图形文件的读取

4.1　彩色图形的储存方式

在 PC 上，常见的彩色图形表示方式有两种:

一、聚集像素式 (Packed Pixel)

二、彩色平面式 (Color Plane)

以 BIOS 的图像模式而言，模式 4、模式 5(320×200 4 色)及模式 13H(320×200 256 色)是属于聚集像素式的表现方法，而其它的绘图模式都是使用彩色平面法来表示. 究竟这两种表示法有什么不同呢? 首先看以下的图示说明:

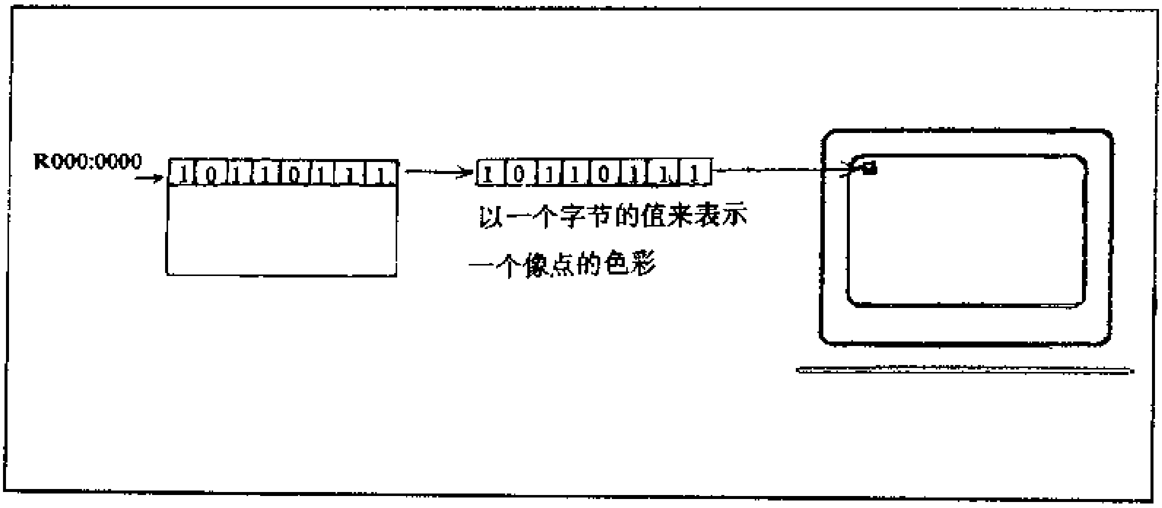


图 4-1　聚集像素表示法 (以 320×200 250 色为例)

由上面两张图可以很清楚的看出，聚集像素表示法是把彩色值搜集在同一个字节内，所以如果整个字节都用来表示一个像素，那么一个字节就可以表示 256 种颜色. 如果 1 字节用来表示 4 个像素 (像模式 4 和 5)，即每个像素以 2 个位来表示，$2^2=4$，可以表示 4 种颜色. 这种色彩表示法的最大好处是直接、简单. 然而这方法却有它的限制，原因如下:

因为 EGA／VGA 的显示内存在 DOS 下的地址为 A0000～AFFFF 共 64K 字节的内存空间，如果以聚集像素法来表示 256 色的绘图模式，则每个字节可以表示一个像点，那么其分辨率仅仅只有 320×200(64000). 如果要表示 16 色的绘图模式，每个字节可以表示 2 个像素，那么其分辨率也不过是 256 色模式的 2 倍，即 640×200 或 320×400，而且只有一页显示页，根本无法满足用户的需求. 这样的分辨率已成为聚集像素的极限，因为地址空间有限，无法无限延伸，所以为了解决这个困境，彩色平面表示法便应运而生了.

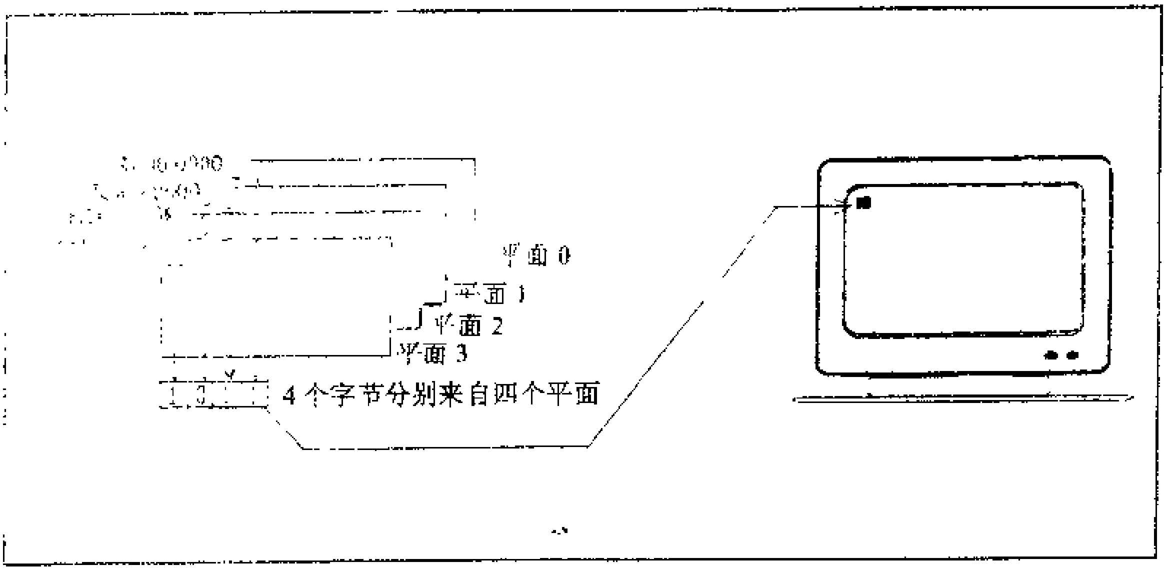


图 4-2　彩色平面表示法

彩色平面表示法是把显示内存以 64K 为单位分成 4 块（以 640×350 16 色为例），分别叫作 plane 0、plane 1，plane 2 及 plane 3，每一个平面控制一种颜色 3（红、蓝、绿及亮度），利用这 4 个平面便可以表示 $2^4=16$ 种彩色。必须特别注意的是，这 4 个平面的地址同为 A0000～AFFFF，也就是说：同样是 A0000:0000 这个地址，却同时指向 4 个字节，那么用户如何知道所存取的是哪一个字节呢？答案是利用 EGA／VGA 所提供的暂存操作来指定。当然，如果利用 BIOS 的读点／写点功能调用，便无需关心彩色平面的问题了；但是，如果决定自行存取的话，彩色平面的问题便不得不注意：彩色平面表示法是 4 个平面上的同一地址上同一位置的 4 个字节（每平面各取 1bit）收集起来，共同表示屏幕上的一个像素（pixel）。

在作图形上有一个必须注意的事项便是调色板的问题。前文所提到的 16 色及 256 色乃是指屏幕同时能呈现的色彩数目，但是屏幕及该适配器的显示能力却不仅于此。例如 EGA 可以显示 64 种不同的颜色，而 VGA 则可以显示高达 262 144 种颜色。然而，由于显示内存的限制，并不能把所有的颜色全部用上。试想，如果在 VGA 下要使用全部的色彩，那么每一个像素就必须用 18 个 bit 来表示（$2^{18}=262\ 144$），不仅在相同分辨下多占 2 倍多的内存，处理速度也会因此跟着减慢。基于内存的考虑，调色板的观念就产生了。

调色板，就是由真正可以用的所有色彩（EGA 有 64 种色彩，VGA 有 262 144 种色彩）中挑出一组（可能 16 种或 256 种，视绘图模式而定）要用的色彩值置于数组中，在 Turbo C 下，预设的调色板（16 色）如下：

表 4-1　Turbo C 预设的调色板定义

颜色名称	色彩下标	实际颜色值
EGA-BLACK	0	0
EGA-BLUE	1	1
EGA-GREEN	2	2

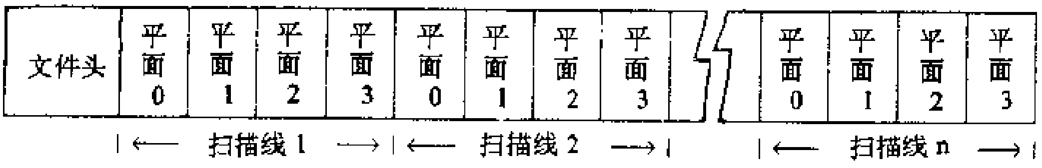
(续表)

颜色名称	色彩下标	实际颜色值
EGA–CYAN	3	3
EGA–RED	4	4
EGA–MAGENTA	5	5
EGA–BROWN	6	20
EGA–LIGHTGRAY	7	7
EGA–DRAKGRAY	8	56
EGA–LIGHTBLUE	9	57
EGA–LIGHTGREEN	10	58
EGA–LIGHTCYAN	11	59
EGA–LIGHTRED	12	60
EGA–LIGHTMAGENTA	13	61
EAG–YELLOW	14	62
EGA–WHITE	15	62

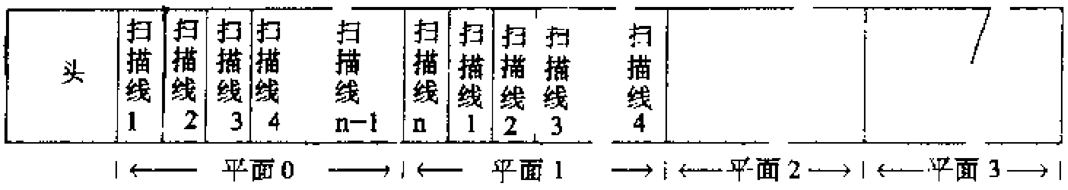
由调色板的概念可以联想到，当我们将一张彩色图形储存于文件中时，通常也会将调色板的值一并存入，所以在读取时便需要将调色板一并读取，并更改屏幕的调色板组合，才能将图形原来的色彩呈现出来。

在图形数据的储存方面，如果是使用聚集像素表示法的话，只是单纯地将图形数据连续储存即可；但如果是色彩平面表示法的话，则可能有如下两种储存方式：

1. 以扫描线为单位：



2. 以彩色平面为单位：



4.2 Turbo C 的 getimage 彩色图形储存格式

在前文曾经提过，彩色图形在主存中或文件中的储存方式，主要有以位为主和以扫描线为主两种。大部分的软件所使用的都是以扫描线为主的储存方式，也就是先储存第一条扫描线的第 0 平面数据、第 1 平面数据、第 2 平面数据及第 3 平面数据，接下来是第二条扫描线的第 0 平面数据……依此类推。Turbo C 也是使用这种方式储存图形数据。

就用户来说，除非遇到了不同于此方式的文件必须转换外，我们仍可粗略地将 Turbo C 的 getimage 彩色图形储存格式看成以下的样子：

宽－1	高 1	彩色图形数据	...		保留
←2bytes→	←2bytes→	. . . .			←2bytes→

对于任何以扫描线为主的文件，只要能改成如上的格式，即可使用 putimage 将图形显示于屏幕。因此，虽然是彩色图形，但是处理步骤却和单色图形没什么差别，基本上只要更改其前面的宽、高成其它的识别字串即可。

在读取彩色的图形文件之时，还有另外一个问题，便是调色板的设定。对于 VGA 所使用的模拟式彩色屏幕而言，理论上可以显示无限多种颜色，然而由于适配器的限制，使得 VGA 所连接的屏幕拥有显示 256K 种颜色的能力。但是在适配器的划分上，凡是要显示于屏幕上的图形数据，都必须先写入相对应的显示内存上。那么，如要在屏幕上能同时显示 256K 种颜色，则每一个像素(pixel)就必须以 18 个位来表示，不仅占内存、而且增加处理的复杂度，也降低了执行速度。所以在 VGA／EGA 等彩色适配器使用了调色板的概念，限制在屏幕上能同时显示的颜色数目，以降低处理的复杂度，其概念如图 4-3 所示：(以 EGA 16 色模式为例)。

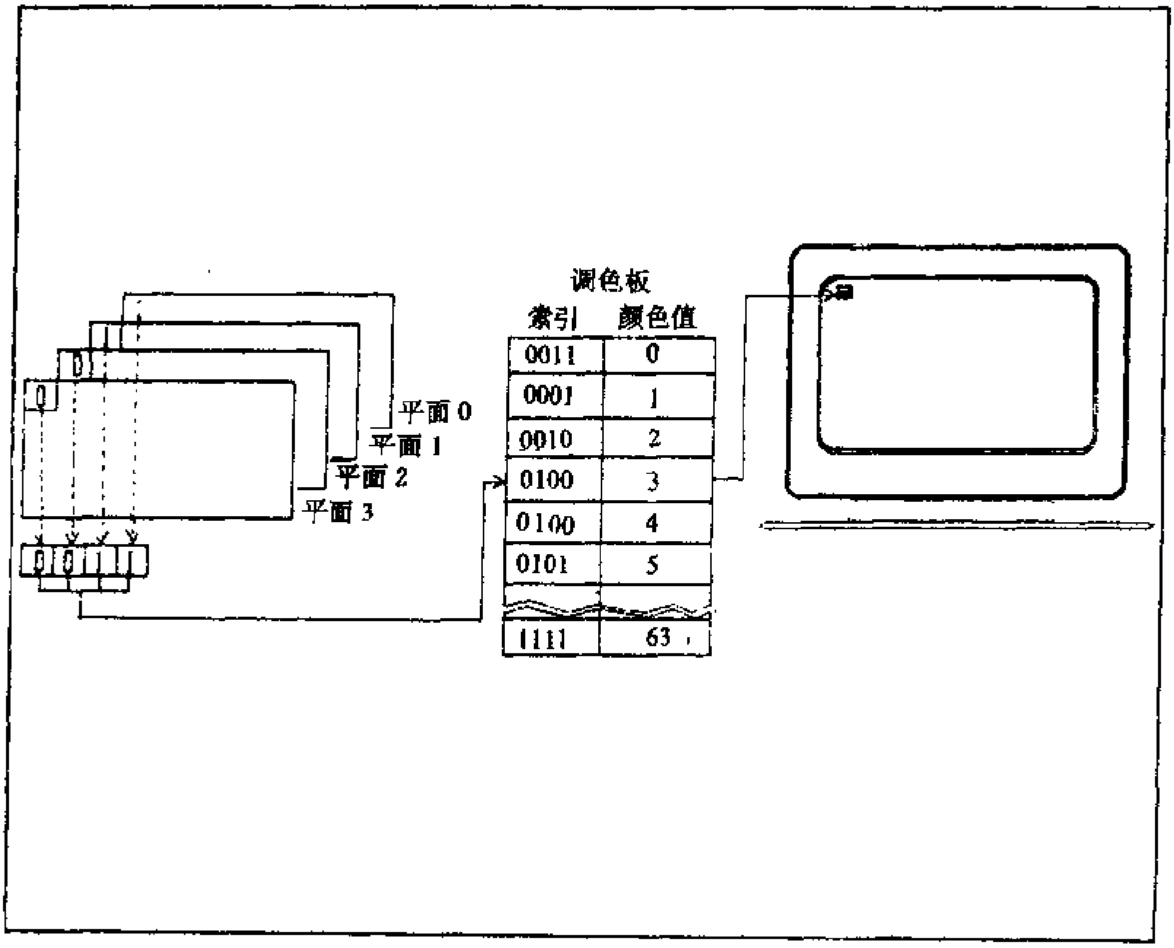


图 4-3 EGA 16 色彩色模式

在内存中所代表的位只是一个下标，通过调色板找到相对应的颜色值再送至屏幕，才是真正出现的颜色。对于 EGA 而言，可以从 64 种颜色中挑出 16 种颜色组成调色板。然而对于 VGA 而言，还要多加一个步骤如下：

如图 4-4 所示，在 VGA 下使用调色板所选到的值并非代表颜色的值，而是色彩寄存器(Color Register)的下标。将彩色寄存器中所记录的三个颜色值输出至屏幕之后，才会得到我们所需要的颜色。由于色彩寄存器中的三个值各以 6 个位来表示，也就是每个值可以有 64 种变化，三个组合起来共有 64 * 64 * 64＝256K，这就是 VGA 可以选择 256K 色的由来。

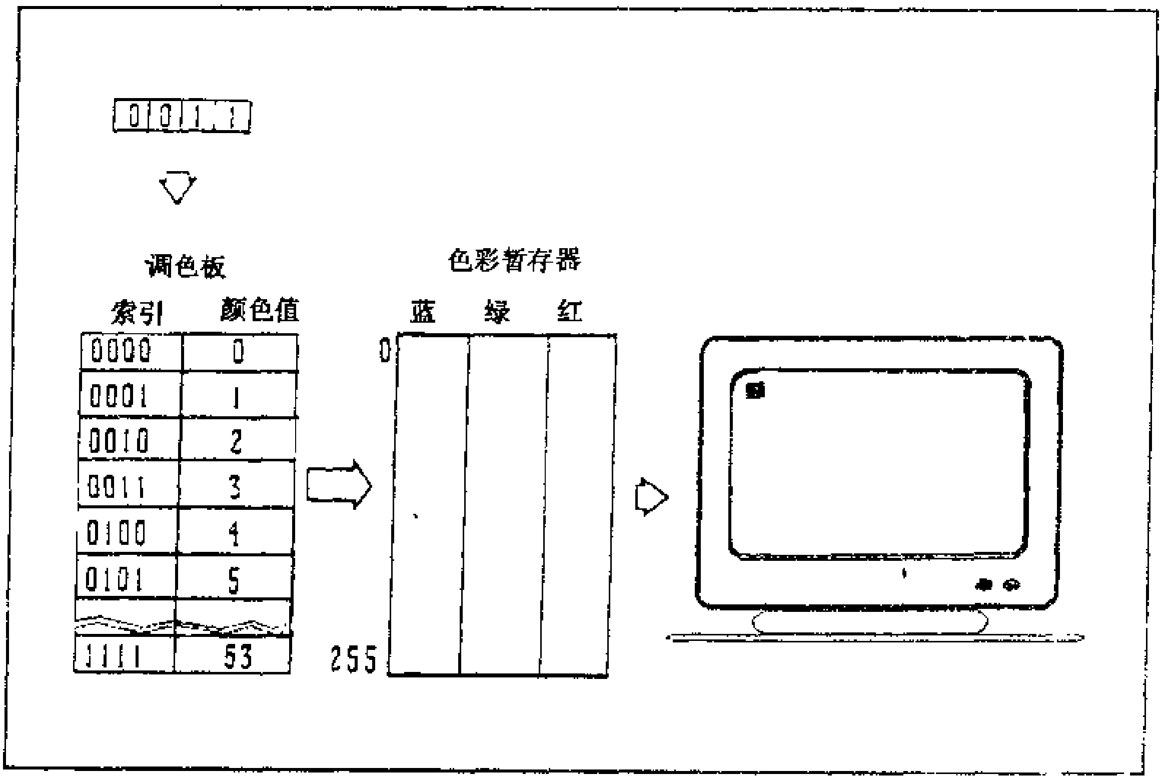


图 4-4　VGA 16 色彩色模式

在常见的几种彩色图形文件中，对于调色板的处理并不相同：

PCX	：	色彩寄存器图形数据同时储存。
TPC、PBC、GPM	：	调色板和图形数据同时储存。
MAP、PUT	：	不储存调色板。

彩色的 MAP 及 PUT 文件本身并没有储存调色板，所以如果你自行以程序将图形显示于屏幕，可能无法呈现原来的颜色。除非在图形输出之前，就已经将调色板设定成和 Rambow Paint 或 Image 72 的文件一样，这点恐怕就只有求助于有储存调色板功能的绘图软件了。绘图大师就具备了这项特点，所以 TPC，BPC 及 GPM 文件都将调色板和图形数据存放在一起，其解读的方法我们将于后文介绍。值得注意的是，笔者手上的绘图大师 1.2 版所储存的是 16 色的调色板，就如前文所提到的，在 VGA 下多了一道色彩寄存器的选择步骤，所以有时候即使我们照着 TPC，BPC 或 GPM 所记录的值来调整系统的

调色板，仍然不能恢复原色，就是因为色彩暂存器不同的原因。解决的方法恐怕只有将16个色彩寄存器全部一并储存一途了。PCX文件就是用这个方法。

到此各位应该是对于彩色图形文件有了更进一步的了解了。就让我们看一个实际的例子吧！

下面读取彩色TPC文件的程序列表，下面几节我们将不使用这种写法，原因也会一并详述。

```
/* CL_TPC.C */

#include <graphics.h>
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <alloc.h>

void load_tpc(int, int, char *, int, int);

main()
{
     int graph_driver=VGA, graph_mode=VGAMED;

     initgraph(&graph_driver, &graph_mode, "");
     load_tpc(0,0,"pmt16.tpc",0,COPY_PUT);
     getch();
     closegraph();
}

void load_tpc(int x,int y,char *filename,int page,int status)
{
   int *buf;
   unsigned size;
   FILE *fp;
   int i;
   struct palettetype pt;

   if((fp = fopen(filename, "rb"))==NULL) {
      closegraph();
      puts("File reading error!");
      exit(1);
   }
   fseek(fp, -16L, SEEK_END);
   fread(pt.colors, 1, 16, fp); /* Read the data of palette */
   pt.size = 16;                /* Maxcolors = 16           */
```

```
    setallpalette(&pt);            /* Set palette              */
    fseek(fp, 0L, SEEK_SET);
    size = filelength(fileno(fp));
    buf = (int *) malloc(size);

    if(buf==NULL) {
        closegraph();
        puts("Memory is not enough!");
        exit(1);
    }
    fread(buf, 1, size, fp);
    fclose(fp);
    setactivepage(page);
    putimage(x,y,buf,status);
    free(buf);
}
```

你是不是已经发现，除了绘图模式的设定以及调色板的设定部分外，其方法是否和单色的读图程序一模一样呢？

4.3 彩色 PUT 图形文件的读取

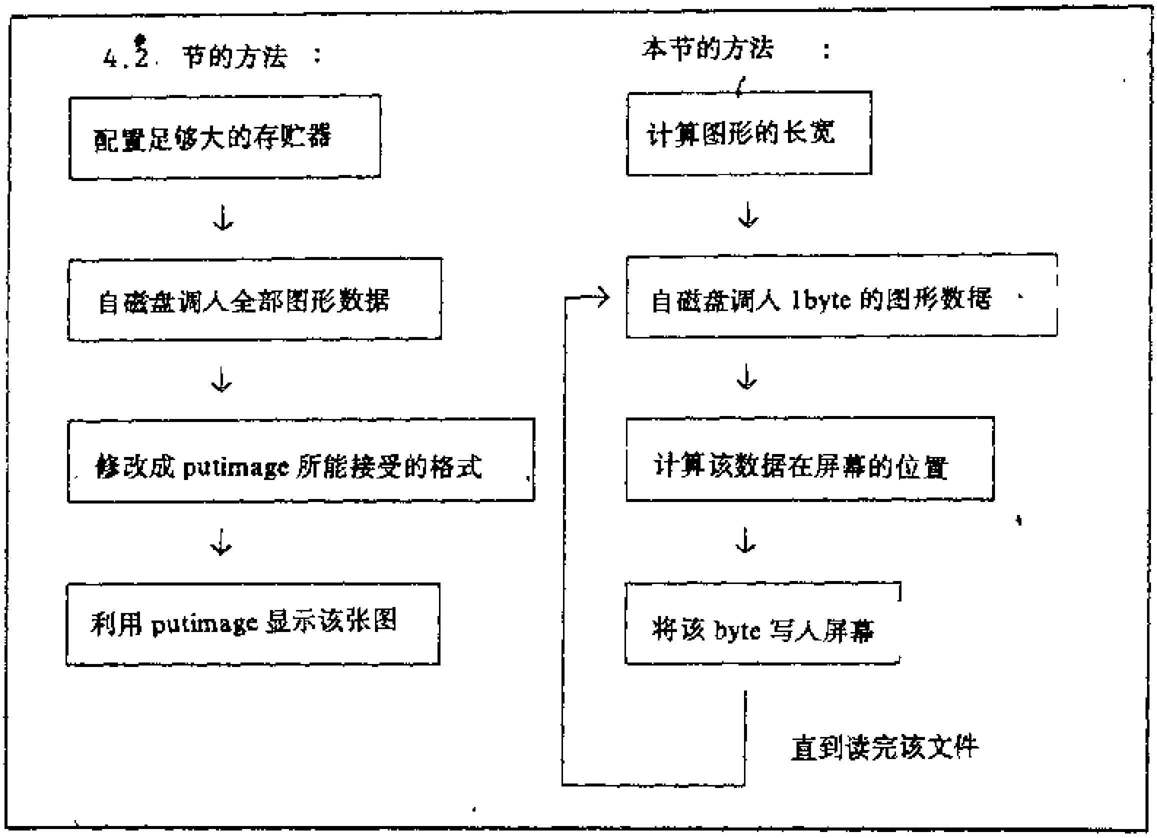


图 4-5 两种彩色图形文件的读取方法的比较

彩色 16 色模式在分辨率 640×350 下储存全页时，需要 112000 bytes，而在 640×480 分辨率下，储存全页所需要的内存更高达 153600 bytes，如果有一个全页或接近全页的 16 色彩色图形文件需要显示的话，使用上一节介绍的方法，则内存的使用将会有捉襟见肘的感叹。所以在以下几节我们将使用另一种方法，完全不需要配置任何内存，不管图形多大都能顺利显示。

不过，有许多事物往往是无法两全其美的，为了节省内存的使用，我们却付出了时间的代价，在硬盘下操作并不会有太大的影响，但是如果是在软盘下操作，数据读取的速度就有些令人无法忍受了。上面是两种方法的比较:

由于本节所介绍的方法一次只读取一个字节数据，所以并无需另行配置额外的内存，也正因为一次只读取一个字节的数据，所以速度才会比较慢，如果你的程序很注重其读取速度的话，建议你不妨将这个程序改为一次以 1 条扫描线为读取单位，相信速度将会有显著的改善，当然这时你就必须适当地配置额外的内存给该扫描线了。以下就是 LADC__PUT.C 的程序列表:

```
/* LADC__PUT.C */

#include <graphics.h>
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <alloc.h>

void load__putc(char *, int);

main()
{
    int graph__driver=VGA, graph__mode=VGAMED;

    initgraph(&graph__driver, &graph__mode, "");
    load__putc("ctitle.put",0);
    getch();
    closegraph();
}
void load__putc(char *filename, int page)
{
   FILE *fp;
   int i, j;
   unsigned lowbyte, highbyte;
   int  x, y;

   if((fp = fopen(filename, "rb"))==NULL) {
      closegraph();
```

```
        puts("File Reading Error!!");
        exit(1);
    }
    lowbyte   = fgetc(fp);
    highbyte = fgetc(fp);
    x = lowbyte + highbyte * 256;
    lowbyte   = fgetc(fp);
    highbyte = fgetc(fp);
    y = lowbyte + highbyte * 256;
    x = (x%8==0) ? x/8 : x/8+1;

    for(i=0;i<y;i++)  {
        outportb(0x3c4, 2); outportb(0x3c5, 1);
        for(j=0;j<x;j++)
            pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));

        outportb(0x3c4, 2); outportb(0x3c5, 2);
        for(j=0;j<x;j++)
            pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));

        outportb(0x3c4, 2); outportb(0x3c5, 4);
        for(j=0;j<x;j++)
            pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));

        outportb(0x3c4, 2); outportb(0x3c5, 8);
        for(j=0;j<x;j++)
            pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));
    }
    outportb(0x3c4, 2); outportb(0x3c5, 0xf);
    fclose(fp);
}
```

在这个程序中，有直接将数据写入显示内存的操作，而由于 EGA／VGA 的 16 色模式有 4 个位平面，所以在写入之前，必须先通知 EGA／VGA 是要将数据写入哪个平面(或哪几个)。而通知的方式则是由写 EGA／VGA 的寄存器来完成，其原理请参考 EGA／VGA 有关书籍，在此仅介绍其方法：

```
outportb(0x3c4,3);      /* 选择对应屏蔽寄存器 */
outportb(0x3c4,c);      /* 指定哪几个平面可以写入 */
```

其中变量 C 由 4 个位组成，由低位到高位分别代表平面 0 至平面 1 可否写入。如果该 bit 为 1，表示可以写入；如果该 bit 为 0，表示不可写入。例如 C＝1 时，平面 0 可写入，而其它平面则不可写入；C＝5 则表示平面 0 及平面 2 可写入，而平面 1 及平面 3 不可写入，依此类推。

LOAC_PUT.C 这个函数首先计算出图形的长、宽，各以 x、y 表示，然后将 x 换算

成以 bytes 为单位，再以扫描线为外循环，一个平面接着一个平面将图形数据存人相对应的位置和平面。在显示完图形后，请记住将全部平面置为使能(enable)状态，同时并关闭文件。

4.4 彩色 MAP 图形文件的读取

MAP 文件的读取除了必须利用

```
for(i=0;i<7;i++)
    fgetc(fp);
```

丢弃前七个 byte 不用，其余和读取 PUT 的方法一样，请直接参考程序 LADC_MAP.C。

```
/* LADC_MAP.C */

#include <graphics.h>
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <alloc.h>

void load_mapc(char *, int);

main()
{
    int graph_driver=VGA, graph_mode=VGAMED;

    initgraph(&graph_driver, &graph_mode, "");
    load_mapc("rambow16.map",0):
    getch();
    closegraph();
}

void load_mapc(char *filename, int page)
{
    FILE *fp;
    int i, j;
    unsigned lowbyte, highbyte;
    int  x, y;
    struct palettetype pt;

if((fp = fopen(filename, "rb"))==NULL) {
    closegraph();
    puts("File Reading Error!!");
```

```
        exit(1);
    }
        for(i=0; i<7; i++)      fgetc(fp);
        lowbyte   = fgetc(fp);
        highbyte  = fgetc(fp);
        x = lowbyte + highbyte * 256;
        lowbyte   = fgetc(fp);
        highbyte  = fgetc(fp);
        y = lowbyte + highbyte * 256;
        x = (x%8==0) ? x/8 : x/8+1;
        for(i=0;i<y;i++)  {
            outportb(0x3c4, 2); outportb(0x3c5, 1);
            for(j=0;j<x;j++)
              pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));

            outportb(0x3c4, 2); outportb(0x3c5, 2);
            for(j=0;j<x;j++)
              pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));

            outportb(0x3c4, 2); outportb(0x3c5, 4);
            for(j=0;j<x;j++)
              pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));

            outportb(0x3c4, 2); outportb(0x3c5, 8);
            for(i=0;j<x;j++)
              pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));
    }
      outportb(0x3c4, 2); outportb(0x3c5, 0xf);
      fclose(fp);
    }
```

4.5 彩色 GPM 图形文件的读取

不同于 PUT 及 MAP，GPM 文件在文件尾部分多记录了该张图形的调色板，其格式如下·

宽	高	图形数据	. . . . . .	'p'	'a'	'l'	't'	
								16个调色盘的值

在实际的应用上，我们可以不理会 palt 4 个调色板识别字母，而直接以如下的方法从文件后面读取调色板的值，并且设定之:

1. 定义变量

2. 移动文件指针至倒数 16 个字节
3. 读取 16 个色码
4. 颜色数目
5. 设定调色板
6. 将文件指针归零

以下就是 LADC_GPM.C 的程序列表:

```
/* LADC_GPM.C */

#include <graphics.h>
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <alloc.h>

void load_gpmc(char *, int);

main()
{
    int graph_driver=VGA, graph_mode=VGAMED:

    initgraph(&graph_driver, &graph_mode, "");
    load_gpmc("dragon16.gpm",0);
    getch();
    closegraph();
}

void load_gpmc(char *filename, int page)
{
     FILE *fp;
     int i, j;
     unsigned lowbyte, highbyte;
     int  x, y;
     struct palettetype pt;

     if((fp = fopen(filename, "rb"))==NULL) {
        closegraph();
        puts("File Reading Error!!");
        exit(1);
}

        fseek(fp,-16L,SEEK_END);
        fread(pt.colors,1,16,fp);
```

```
37:      pt.size = 16;
38:      setallpalette(&pt);           /* Set the palette */
39:      fseekfp.0l,SEEK_SET);
40:      lowbyte = fgetc(fp);
41:      highbyte = fgetc(fp);
42:    x = lowbyte + highbyte * 256;
43:    lowbyte  = fgetc(fp);
44:   highbyte = fgetc(fp);
45:   y = lowbyte + highbyte * 256;
46:    x = (x%8 = =0) ? x / 8 : x / 8+1;
47:
48:    for(i=0;i<y;i++)  {
49:      outportb(0x3c4, 2); outportb(0x3c5, 1);
50:      for(j=0;j<x;j++)
51:         pokeb(((page) ? 0xa800 : 0xa000),i * 80+j,fgetc(fp));
52:
53:      outportb(0x3c4, 2); outportb(0x3c5, 2);
54:      for(j=0;j<x;j++)
55:         pokeb(((page) ? 0xa800 : 0xa000),i * 80+j,fgetc(fp));
56:
57:      outportb(0x3c4, 2); outportb(0x3c5, 4);
58:      for(j=0;j<x;j++)
59:         pokeb(((page) ? 0xa800 : 0xa000),i * 80+j,fgetc(fp));
60:
61:      outportb(0x3c4, 2); outportb(0x3c5, 8);
62:      for(j=0;j<x;j++)
63:         pokeb(((page) ? 0xa800 : 0xa000),i * 80+j,fgetc(fp));
64: }
65;      outportb(0x3c4, 2); outportb(0x3c5, 0xf);
66;      fclose(fp);
67:}
```

程序说明:

11-19	主程序
23-27	定义变量
35	移动文件指针至倒数16个字节
36	读取16个色码
37	设定颜色数目
38	设定调色板 setallpalette(&pt);
39	将文件指针归零fseek(fp,OL,SEEK-SET);
48-64	读写调色板，显示图形

4.6 彩色 TPC 图形文件的读取

由于 TPC 文件前端所记载的是宽减 1 及高减 1，所以在计算完 x 及 y 值之时，请务必再各加上 1，否则将会漏失一个图形数据。彩色 TPC 图形文件的读取程序如下:

```
/* LADC_TPC.C */

#include <graphics.h>
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <alloc.h>

void load_tpcc(char *, int);

main()
{
    int graph_driver=VGA, graph_mode=VGAMED;

    initgraph(&graph_driver, &graph_mode, "");
    load_tpcc("pmt16.tpc",0);
    getch();
    closegraph();
}

void load_tpcc(char *filename, int page)
{
    FILE *fp;
    int i, j;
    unsigned lowbyte, highbyte;
    int  x, y;
    struct palettetype pt;

    if((fp = fopen(filename, "rb"))==NULL) {
        closegraph();
        puts("File Reading Error!!");
        exit(1);
}

    fseek(fp, -16L, SEEK_END);
    fread(pt.colors, 1, 16, fp);
    pt.size = 16;
    setallpalette(&pt);                    /* Set the palette */
```

```
    fseek(fp, 0L, SEEK_SET);
    lowbyte  = fgetc(fp);
    highbyte = fgetc(fp);
    x = lowbyte + highbyte * 256;
    x ++;
    lowbyte  = fgetc(fp);
    highbyte = fgetc(fp);
    y = lowbyte + highbyte * 256;
    y ++;
    x = (x%8==0) ? x/8 : x/8+1;

    for(i=0;i<y;i++) {
        outportb(0x3c4, 2); outportb(0x3c5, 1);
        for(j=0;j<x;j++)
            pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));

        outportb(0x3c4, 2); outportb(0x3c5, 2);
        for(j=0;j<x;j++)
            pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));

        outportb(0x3c4, 2); outportb(0x3c5, 4);
        for(j=0;j<x;j++)
            pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));

        outportb(0x3c4, 2); outportb(0x3c5, 8);
        for(j=0;j<x;j++)
            pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));
}
      outportb(0x3c4, 2); outportb(0x3c5, 0xf);
      fclose(fp);
}
```

4.7 彩色 BPC 图形文件的读取

请直接参考程序 LADC_BPC.C:

```
/* LADC_BPC.C */

#include <graphics.h>
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <alloc.h>
```

```
void load_bpcc(char *, int);

main()
{
    int graph_driver=VGA, graph_mode=VGAMED;

    initgraph(&graph_driver, &graph_mode, "");
    load_bpcc("pmt16.bpc",0);
    getch();
    closegraph();
}

void load_bpcc(char *filename, int page)

    FILE *fp;
    int i, j;
    unsigned lowbyte, highbyte;
    int  x, y;
    struct palettetype pt;

if((fp = fopen(filename, "rb"))==NULL) {
      closegraph();
      puts("File Reading Error!!");
      exit(1);
}

      fseek(fp, -16L, SEEK_END);
      fread(pt.colors, 1, 16, fp);
      pt.size = 16;
      setallpalette(&pt);            /* Set the palette */
      fseek(fp, 0L, SEEK_SET);
      for(i=0; i<7; i++)    fgetc(fp);
      lowbyte  = fgetc(fp);
      highbyte = fgetc(fp);
      x = lowbyte + highbyte * 256;
      lowbyte  = fgetc(fp);
      highbyte = fgetc(fp);
      y = lowbyte + highbyte * 256;
      x = (x%8==0) ? x/8 : x/8+1;

      for(i=0;i<y;i++)  {
      outportb(0x3c4, 2); outportb(0x3c5, 1);
      for(j=0;j<x;j++)
         pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));
```

```
        outportb(0x3c4, 2); outportb(0x3c5, 2);
        for(j=0;j<x;j++)
            pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));

        outportb(0x3c4, 2); outportb(0x3c5, 4);
        for(j=0;j<x;j++)
            pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));

        outportb(0x3c4, 2); outportb(0x3c5, 8);
        for(j=0;j<x;j++)
            pokeb(((page) ? 0xa800 : 0xa000),i*80+j,fgetc(fp));
}
    outportb(0x3c4, 2); outportb(0x3c5, 0xf);
    fclose(fp);
}
```

4.8 彩色 PCX 图形文件的读取

PCX 图形文件的格式请参考第 3.2.7 节，而彩色 PCX 图形文件的读取则可以由下列程序完成:

```
/* LADC_PCX.C */

#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>

typedef struct {
        char manufacturer;
        char version;
        char encoding;
        char bits_per_pixel;
        int xmin, ymin;
        int xmax, ymax;
        int hres;
        int vres;
        char palette[48];
        char reserved;
        char colour_planes;
        int bytes_per_line;
```

```
        int palette_type;
        char filler[58];
        } PCXHEAD;

void load_pcxc(char *, int);

main()
{
    int graph_driver=VGA, graph_mode=VGAMED;

    initgraph(&graph_driver, &graph_mode, "");
    load_pcxc("demo.pcx", 0);
    getch();
    closegraph();
}

void load_pcxc(char *filename, int page)
{
    PCXHEAD header;
    FILE *fp;
    register int i, j;
    int cnt=0, total, c;
    char picdata;
    unsigned int x, y, bx;

    if((fp=fopen(filename, "rb"))==NULL)    {
        closegraph();
        puts("File reading error");
        exit(1);
    }
    fread( (char *) &header, 1, sizeof(PCXHEAD), fp);
    x = header.xmax - header.xmin + 1;
    y = header.ymax - header.ymin + 1;
    y = (y <= getmaxy()) ? y : getmaxy()+1;
    bx = (x%8) ? (x/8)+1 : x/8;

    for(i=0; i<y; i++)      {
        for(c=0; c<4; c++)      {
            total = 0;
            while(total < bx)   {
                cnt = 1;
                picdata = fgetc(fp);
                if(0xc0==(0xc0 & picdata))  {
                    cnt = 0x3f & picdata;
```

```
                        picdata = fgetc(fp);
                    }
                    for(j=total; j<total + cnt; j++) {
                        outportb(0x3c4, 2);
                        outportb(0x3c5, 1 << c);
                        pokeb( (page) ? 0xa800 : 0xa000,
                              i * 80+j, picdata);
                    }
                    total += cnt;
                }
            }
        }
        fclose(fp);
    }
```

图 4-6　程序 LADC_PCX.C 的执行实例

```
                    picdata = fgetc(fp);
                }
                for(j=total; j<total + cnt; j++) {
                    outportb(0x3c4, 2);
                    outportb(0x3c5, 1 << c);
                    pokeb( (page) ? 0xa800 : 0xa000,
                        i * 80+j, picdata);
                }
                total += cnt;
            }
        }
    }
    fclose(fp);
}
```

图 4-6 程序 LADC_PCX.C 的执行实例

第五章　Super VGA 绘图程序设计

5.1　Super VGA 简介

随着计算机科技的开发，个人计算机的应用范围逐渐广泛了，众多的用户对于计算机及相关外围设备的要求也愈来愈高．以屏幕及显示适配器而言，虽然 IBM 推出了增强型彩色显示卡 EGA，紧接着 VGA 也成为工业界彩色显示卡的标准，然而其分辨率及色彩仍然无法满足用户的需求．于是各家制造彩色显示卡的厂商便纷纷以原来的 VGA 卡为基础，提高其分辨率，并扩充了若干功能，使得该适配器除了拥有原来的 VGA 所有的绘图模式之外，还增加了许多更强的功能及更高的分辨率，这便是所谓的 Super VGA．

一般而言，Super VGA 由于是建构在 VGA 的标准上，所以我们可以把它看成是如下面的样子：

Super VGA　=　VGA　+　扩充功能

由于 VGA 目前已经是工业界的标准，所以凡是在 VGA 下所写的软件，均可以在 Super VGA 上执行，而在 Super VGA 下开发的软件只要没有使用到扩充的功能，也可以在任何一种 Super VGA 或 VGA 下执行；但是若在程序中使用该 Super VGA 的扩充部分，这种通用性的便利就无法继续存在了．正如前面所提到过的：Super VGA 是各制造厂所自行设计的，其间并没有订立标准，所以虽然各家所提供的功能大同小异，然而其调用的方法和寄存器的地址可能不尽相同，这是我们在 Super VGA 下设计程序所必须要注意的地方．

在正式讨论 Super VGA 的程序设计之前，先让我们来看看 Super VGA 到底提供了哪些扩充的功能：

首先，当然是更高的分辨率．在前文介绍彩色绘图时，曾经提到 PC 上目前通用的彩色绘图模式基本上可以分成 16 色及 256 色两大类．在 VGA 下，16 色模式最高可达 640×480，而 256 色模式的分辨率则只有 320×200．在 Super VGA 下，16 色及 256 色模式的分辨率则可为 800×600，1024×768 甚至更高．当然分辨率的高低及色彩的多少也取决于显示内存的大小，尽管 Super VGA 有显示高分辨率及多色彩的能力，若显示内存不足，也无法将该 Super VGA 发挥到极致，所以想要知道你所拥有的适配器究竟有多少功能，除了看看该适配器是使用哪一种芯片外，还有就是该适配器拥有多少显示内存了．

另外，有些制造商的 Super VGA 芯片还提供了硬件放大的功能．以往想要在屏幕上将图形放大只有使用软件的方法，经过程序的运算再将结果放到屏幕，由于软件的速度远不及硬件，所以产生的效果相当有限，速度又不尽如人意；而今，这个问题却由 Super VGA 上芯片的硬件线路帮你解决了．

当然，由于各家芯片各不相同，所提供的功能也互有差异，无法在此一一详述，读者如果有兴趣的话可参阅该适配器的使用手册，先确定是属于哪一种芯片。不同的适配器制造商也有可能使用相同的芯片，而该适配器的能力主要也取决于芯片的种类，型号及视频 BIOS 的版本，以及拥有多少 K 的显示内存。

至于如何判断所拥有的适配器呢？除了查阅手册以外最简单的方法就是利用 DEBUG 这个工具程序看看 C000:0000 的地址，通常 Super VGA 的制造商有关信息都会放在这里。以下就是 Trident Super VGA 的例子：

```
c>debug
-dc000:0000
c000:0000  55 AA 40 EB 48 37 34 30-30 30 37 7A 78 2A 2A 20   U.@.H740007zx**
c000:0010  20 52 45 53 45 52 56 45-44 20 46 4F 52 20 49 42   REVERVED FOR IB
c000:0020  4D 20 43 4F 4D 50 41 54-49 42 49 4C 49 54 59 20   M CMPATIBILITY
c000:0030  2A 2A 69 A5 00 00 90 10-30 31 2F 32 32 2F 39 31   **i.....01/22/91
c000:0040  00 00 20 43 32 2E 31 20-20 20 20 20 20 E9 75 10   ..C2.1        .u.
c000:0050  07 43 6F 70 79 72 69 67-68 74 20 31 39 38 38 2D   .Copyright 1988-
c000:0060  31 39 39 30 20 54 52 49-44 45 4E 54 20 4D 49 43   1990 TRIDENT MIC
c000:0070  52 4F 53 59 53 54 45 4D-53 20 49 4E 43 2E 0D 0A   ROSYSTEMS INC...
-d
c000:0080  00 07 07 43 6F 70 79 72-69 67 68 74 20 31 39 38   ...Copyright 19s
c000:0090  37 2D 31 39 39 30 20 51-75 61 64 74 64 6C 20 43   7-1990 Quadtel C
c000:00A0  6F 72 70 2E 0D 0A 00 01-00 00 0E 20 32 35 36 00   orp........ 256.
c000:00B0  0E 20 35 31 32 00 0E 20-37 36 38 00 0E 31 30 32   . 512.. 768..102
c000:00C0  34 00 07 4B 20 20 00 0E-56 47 41 00 0E 45 47 41   4..K  ..VGA..EGA
c000:00D0  00 0E 43 47 41 00 0E 4D-44 41 00 07 20 4D 4F 44   ..CGA..MDA.. MOD
c000:00E0  45 0D 0A 00 F6 C7 10 74-06 BE 5A 78 E8 A9 3B F6   E......t..Zx..;.
c000:00F0  C7 20 74 3B E8 BE 01 25-03 00 B2 06 F6 E2 BE AA   . t;...%........
-
```

下面则是 Tseng（俗称曾氏卡）Super VGA 的例子：

```
[Richard Ho] C:\>debug
-dc000:0000
C000:0000  55 AA 30 EB 5B 54 68 69-73 20 69 73 20 6E 6F 74   U.0.[This is not
C000:0010  20 61 20 70 72 6F 64 75-63 74 20 6F 66 20 49 42    a product of IB
C000:0020  4D 20 20 28 49 42 4D 20-69 73 20 61 20 74 72 61   M (IBM is a tra
C000:0030  64 65 6D 61 72 6B 20 6F-66 20 49 6E 74 65 72 6E   demark of Intern
C000:0040  61 74 69 6F 6E 61 6C 20-42 75 73 69 6E 65 73 73   ational Business
C000:0050  20 4D 61 63 68 69 6E 65-73 20 43 6F 72 70 2E 29    Machines Corp.)
C000:0060  EB 59 20 2A 20 43 6F 70-79 72 69 67 68 74 28 63   .Y * Copyright(c
C000:0070  29 31 39 38 38 20 54 73-65 6E 67 20 4C 61 62 6F   )1988 Tseng Labo
-d
```

```
C000:0080  72 61 74 6F 72 69 65 73-2C 20 49 6E 63 2E 20 30   ratories, Inc. 0
C000:0090  32 2F 32 33 2F 38 39 20-56 38 2E 30 30 58 01 19   2/23/89 V8.00x..
C000:00A0  50 00 C0 00 00 00 00 00-00 00 00 00 00 00 00 BD   P...............
C000:00B0  00 C0 00 00 00 00 00 00-00 00 00 EB 3E 1A 00 D7   ...........>...
C000:00C0  00 00 C0 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
C000:00D0  00 00 00 00 00 00 00 10-01 08 00 00 00 00 01 00   ................
C000:00E0  02 02 01 00 04 04 01 00-05 02 05 00 06 01 06 05   ................
C000:00F0  06 00 08 01 08 00 07 02-07 06 07 BA B4 03 B8 00   ................
-
```

由于国内使用曾氏卡的人数较多，而曾氏卡也的确具有较强、较完备的功能，因此本章将讨论的重点放在曾氏 ET-3000 上。如果你使用的是其它厂牌的适配器，那么也许本章所提供的程序则无法在你的计算机上正确地运行，不过，书上所描述的原理仍然值得参考。请你在详细阅读本书说明的原理后再翻阅使用手册，更改其相关部分（如 BIOS 的参数，I／O 口的地址或寄存器的编号），相信应该可以顺利执行。

5.2 Super VGA 的结构

Super VGA 显示器的硬件结构非常复杂，限于篇幅，本节不打算针对它作深入的探讨，仅就程序设计的观点，提供在 Super VGA 上从事程序设计所应具备的常识。

对程序设计者而言，Super VGA 上的内存映射及配置方式是最令人关心的部分，在此我们先来看看 PC 在 DOS 下整个内存是如何划分的：

表 5-1 PC 的内存使用状况

0000:0000	中断向量表
0000:0400	BIOS 数据区
	DOS 的核心部分
	磁盘缓冲区、设备驱动程序…
	COMMAND.COM 的驻留部分
	用户驻留程序
	用户空间
	COMMAND.COM 的非驻留部分
A000:0000	VGA／EGA 显示内存
B000:0000	MDA／VGA／EGA 显示内存，也为 Hercules 显示内存的第 0 页．
B800:0000	CGA／VGA／EGA 显示内存，也为 Hercules 显示内存的第 1 页．
C000:0000	VGA／EGA ROM BIOS
C800:0000	XT DISK BIOS
CC00:0000	其它适配器的 BIOS
E000:0000	LAN,EMS
F000:0000	SYSTEM ROM BIOS
FFFO:FFFF	

其中，VGA / Super VGA 所使用到的内存地址又可详列如下：

A000:0000～B000:FFFF	扩展的图形数据储存地址，然而大部分的模式仍然只使用到 A000:FFFF。
A000:0000～A000:FFFF	标准VGA图形数据的储存空间，(MOD D,E,F,…)。
B000:0000～B000:7FFF	单色文本模式（MODE 7），Hercules绘图模式第0页。
B800:0000～B800:7FFF	单色文本模式（MODE 7），Hercules绘图模式第1页。
C000:0000～C000:5FFF	IBM VGA ROM BIOS（大部分的VGA使用到C000:7FFF)。

由以上的内存配置可以发现，图形数据所能储存的最大空间为 A000:0000～B000:FFFF，为单彩色文本模式下数据存储的地址，以及 Hercules 单色绘图适配器图形数据的储存空间。大部分的 VGA / Super VGA 为了顾及相容性，以便让一部 PC 可同时连接单彩色屏幕起见，通常只使用到 A000:0000～A000:FFFF 共 64KB 的空间。但是 64KB 最多能提供多高的分辨率及多少颜色呢？就 16 色而言，每 4 个 bit 表示一个像素，则最大分辨率为 320×400 或 640×200，只有一页视频页；如果想要显示 256 色的话，那么最高分辨率就只剩下 320×200，且只有一页视频页了。当然这种分辨率根本不能符合用户的需求，所以在有限的可寻址空间下又想要使用更多的显示内存，唯有引进虚拟内存的分页及分段的观念与技术，配合位平面的结构才能完成.

5.2.1 位平面法

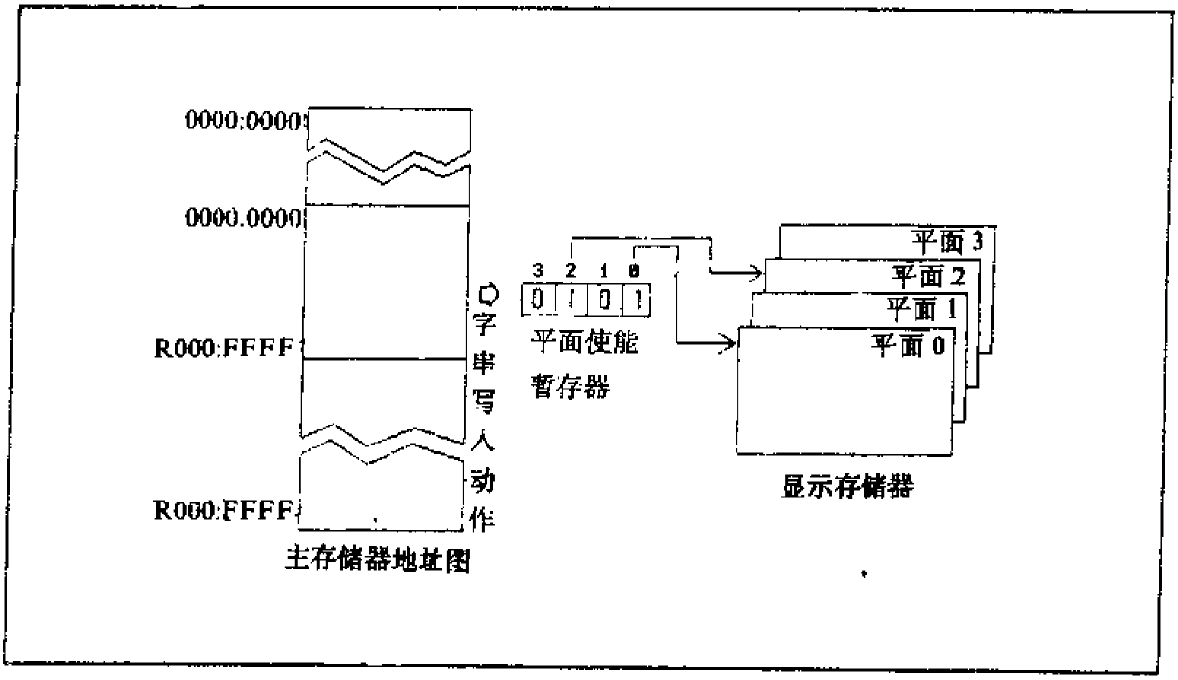


图 5-1 位平面的内存地址图

位平面（bit plane）的作法在 IBM EGA / VGA 上就已经开始采用了，这个方法将

显示内存分成四个部分对应到从 A000:0000 开始的 640KB 地址内，则共可使用到 256KB 的内存（64KB×4=256KB），足以供应大部分 16 色绘图模式的需要。位平面结构可以用图 5−1 来表示：

在上图中，4 块显示内存的地址全部对应到 A000:0000～A000:FFFF 上，也就是说 A000:0000～A000:FFFF 之间的任一地址可同时指向 4 个储存单位，在执行写入或读出时，再由平面使能寄存器决定是读写哪些平面上的数据。当然上图简化了许多的处理程序，较详细的图解请参考有关的书籍。

5.2.2 分页法

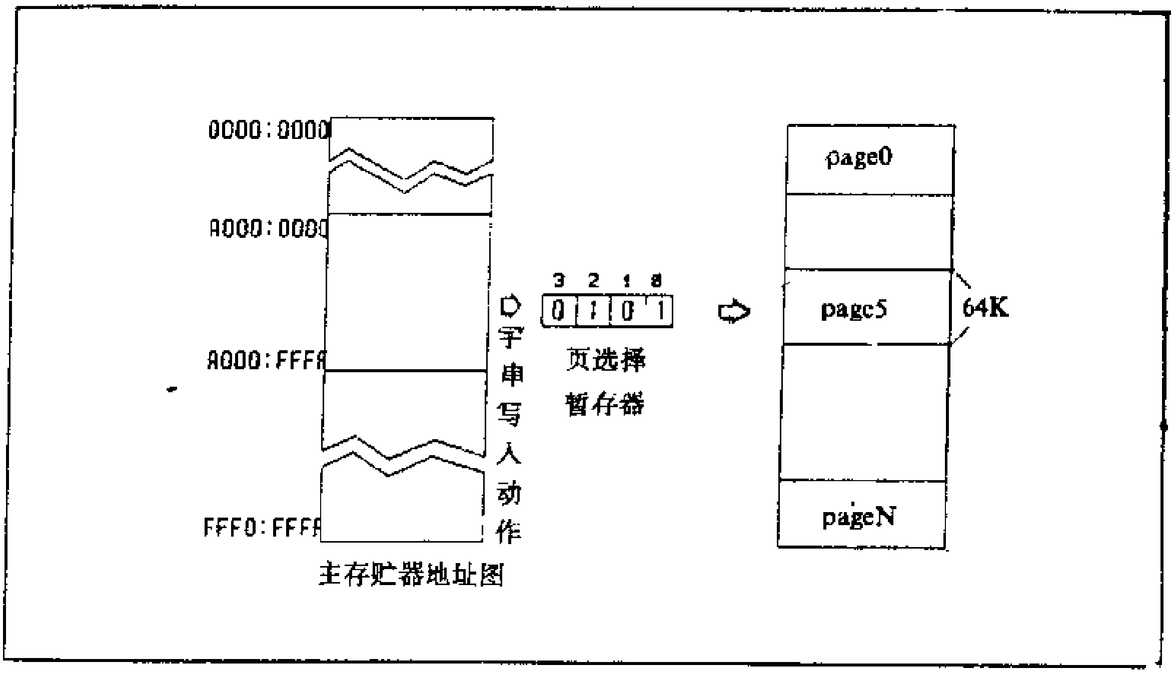


图 5−2　分页法的内存地址图

分页的作法和位平面法最大的不同点在于分页法的一个地址同时只对应到 1 个储存单位。基本上，A000:0000～A000:FFFF 就好像是一个 64K 大小的窗口，程序设计者必须通过这个窗口来存取显示内存，而适配器再直接从显示内存将图形数据显示于屏幕上。页选择寄存器提供程序设计者自由选择所要存取的数据所在页的能力，至于显示内存最多分成几页则视绘图模式而定。

如果我们要存取一个图形数据，那么除了要指定该图形数据的段(segment)地址及偏移(offset)地址外，还要指明其所在的页数。例如：

page:segment:offset

5.3　使用 BIOS 常用程序绘图

5.3.1 Super VGA BIOS 简介

许多硬件制造商为了顾及用户的方便性及硬件的兼容性，都会为他们的新产品撰写一

个 BIOS 程序固化在 ROM 上，提供用户调用使用，以避免用户直接触及复杂的内部结构，而增加无谓的困难。Super VGA 也是一样，大部分的 Super VGA 都有一个和 VGA 兼容的 BIOS 放在地址 C000:0000～C000:5FFF(或 C000:7FFF)的地方。许多的 Super VGA 的扩充功能都是直接附加在原来的 VGA BIOS 的功能调用上。例如原来 VGA BIOS 的 0 号功能（AH＝0）是设定显示模式，原来的标准 VGA 只有 0～13h 等模式可供选择，而 Super VGA 则增加 22h，2Eh，3Dh，29h，37h 等增强模式（以曾氏卡为例），同时其它的有关功能如读、写像素点等在这些增强模式下也可使用。程序设计者通过 BIOS 来绘图根本无须考虑 Super VGA 的硬件结构，何时需要换页，因为这些都由 BIOS 上的常用程序解决了。程序员所必须做的，只要将要写入的像素点的逻辑坐标值告诉 BIOS 就行了。

然而利用 BIOS 来绘图却有一个致命的缺点：速度太慢。如果你很注重绘图的速度的话，那恐怕得自行开发基本的绘点、绘线常用程序了。下一节我们将会介绍如何去编制这样的常用程序，不过在本节我们仍然利用 BIOS 来开发几个有用的绘图常用程序。

5.3.2 选择一个绘图模式

先看看 BIOS 的第 0 号功能调用：

功能0:　设定显示模式

参数:　AH＝0

　　　AL＝显示模式

返回值:　无

调用例:

```
union REGS r;
r.h.ah=0;
r.h.al=0x2e: / * set 640 x 480-256 mode * /
int86(0x10;&r,&r):
```

通过 BIOS 第 0 号功能我们可以指定一个所需要的绘图模式。究竟有哪些增强的绘图模式呢？以笔者现有的 VGA－Ⅲ为例（使用 ET3000－AX Video ASIC Chip），共有下列的增强绘图模式：

表 5-2 ET 3000 的增强绘图模式

Enhanced Graphics Mode	Buffer Start	Max. Page	Display Size	Color
29	A000	1	800×600	16
2D	A000	1	640×350	256
2E	A000	1	640×480	256
30	A000	1	800×600	256
37	A000	1	1024×768	16

不是使用 VGA-Ⅲ的读者，请自行参考该 Super VGA 卡的手册，找出相对应的模式。

为了便于记忆且易于理解，在此我们先定义一个头文件(HeaderFile)SuperVGA.H 如下:

```
/*********************************/
/*        Header file of Super     VGA         */
/*            SUPERVGA.H                       */
/*********************************/
#define GC16_800x600     0x29
#define GC16_1024x768    0x37
#define GC256_640x350    0x2d
#define GC256_640x480    0x2e
#define GC256_800x600    0x30
#define TEXT_MODE        0x03
```

接着我们就可以写一个设定绘图模式的测试程序，并利用这些常数来调用看看，下面就是 SETMODE.C 程序列表:

```
#include <dos.h>
#include <stdio.h>
#include <bios.h>
#include "supervga.h"

void Setvideomode(int);

main()
{
    Setvideomode(GC256_800x600);
    printf("Now, in 800x600 256-color mode\n");
    getch();
    Setvideomode(TEXT_MODE);
}

void Setvideomode(int mode)
{
    union REGS r;

    r.h.ah = 0;
    r.h.al = mode;
    int86(0x10, &r, &r);
}
```

这个程序很短很容易理解。Printf 函数由于是通过 BIOS 的调用来写入字串的，所以在此使用本函数，执行后会先设定为 800×600 256 色模式，然后在左上角显示“Now,in

800x600 256—Color mode"字样，等按下任意键后再恢复成彩色文本模式(mode 3)。

在此再强调一次，如果读者并非使用 VGA-Ⅲ适配器的话，请参考使用手册更改 SUPERVGA.H 中的常数后面的数值，则本程序(SETMODE.C)即可执行无误。

5.3.3 利用 BIOS 功能调用读点写点

设定了合适的绘图模式之后，接着让我们来看看如何使用 BIOS 的功能调用来读写一个像素点：

功能0C	写入一个像素 (Pixel)
参数:	Ah＝0C
	AL＝颜色值(0～maxcolor－1)
	BH＝写入页码
	CX＝X坐标
	DX＝Y坐标
返回值:	无
调用例:	在(200,100)的坐标处写入15的颜色值，并指定写入第0页。

```
union REGS r;

r.h.al  =  15;
r.h.ah  =  0x0c;
r.h.bh  =  0;
r.x.cx  =  200;
r.x.dx  =  100;
int86(0x10,&r,&r);
```

功能0D	读取第一个像素的值
参数:	Ah＝0D
	BH＝读取像素的页码
	CX＝X坐标
	DX＝Y坐标
返回值:	AL＝颜色值
调用例:	读取在(200,100)处的像素的颜色值返回。

```
union REGS r;
    r.h.ah  =  0x0d;
    r.h.bh  =  0;
    r.x.cx  =  200;
    r.x.dx  =  100;
    int86(0x10,&r,&r);
```

应用以上两个功能调用，我们可以写几个简单的小程序来测试 BIOS 的写点及读点的功能。

第一个程序"天女散花"，是利用随机函数 randomize 及 random 产生适当随机数，然

后再将对应的像素点值写入该地址，直到按下任意键即结束本程序。其列表如下:

```
#include <dos.h>
#include <stdio.h>
#include <bios.h>
#include <time.h>
#include <stdlib.h>
#include "supervga.h"

void Setvideomode(int);
void Putpixel(int, int, int);

main()
{
    int x, y;      /* the coordinate of pixel */
    int color;     /* the color of pixel */

    Setvideomode(GC256_800x600);
    randomize();
    do {
        x = random(800);
        y = random(600);
        color = random(256);
        Putpixel(x, y, color);
        } while(!kbhit());
        Setvideomode(TEXT_MODE);
}

void Setvideomode(int mode)
{
    union REGS r;

    r.h.ah = 0;
    r.h.al = mode;
    int86(0x10, &r, &r);
}

void Putpixel(int x, int y, int color)
{
    union REGS r;

    r.h.ah = 0x0c;
    r.h.al = color;
    r.h.bh = 0;
```

```
    r.x.cx  =  x;
    r.x.dx  =  y;
    int86(0x10, &r, &r);
}
```

第二个程序"区块倾斜"，其功能是将指定区块(sx,sy)-(ex,ey)内的图形数据倾斜一固定角度后画在所指定的坐标(dx,dy)处，请看程序 SLANT.C：

```
#include <dos.h>
#include <stdio.h>
#include <bios.h>
#include "supervga.h"

void Setvideomode(int);
void Putpixel(int, int, int);
int  Getpixel(int, int);
void Slant(int, int, int, int, int, int);

main()
{
    Setvideomode(GC256_640x350);
    printf("Slanting character test\n");
    Slant(0,0,182,14,0,200);
    getch();
    Setvideomode(TEXT_MODE);
}

void Setvideomode(int mode)
{
    union REGS r;

    r.h.ah  =  0;
    r.h.al  =  mode;
    int86(0x10, &r, &r);
}

void Putpixel(int x, int y, int color)
{
    union REGS r;

    r.h.ah  =  0x0c;
    r.h.al  =  color;
    r.h.bh  =  0;
    r.x.cx  =  x;
    r.x.dx  =  y;
```

```
    int86(0x10, &r, &r);
}

int  Getpixel(int x, int y)
{
    union REGS r;
    r.h.ah = 0x0d;
    r.h.bh = 0;
    r.x.cx = x;
    r.x.dx = y:
    int86(0x10, &r, &r);
    return r.h.al;
}
void Slant(int sx, int sy, int ex, int ey, int dx, int dy)
{
    int x, y;
    for(y=sy; y<ey; y++)
       for(x=sx; x<ex; x++)
      Putpixel(dx+x+(14-y), dy+y, Getpixel(x, y));
}
```

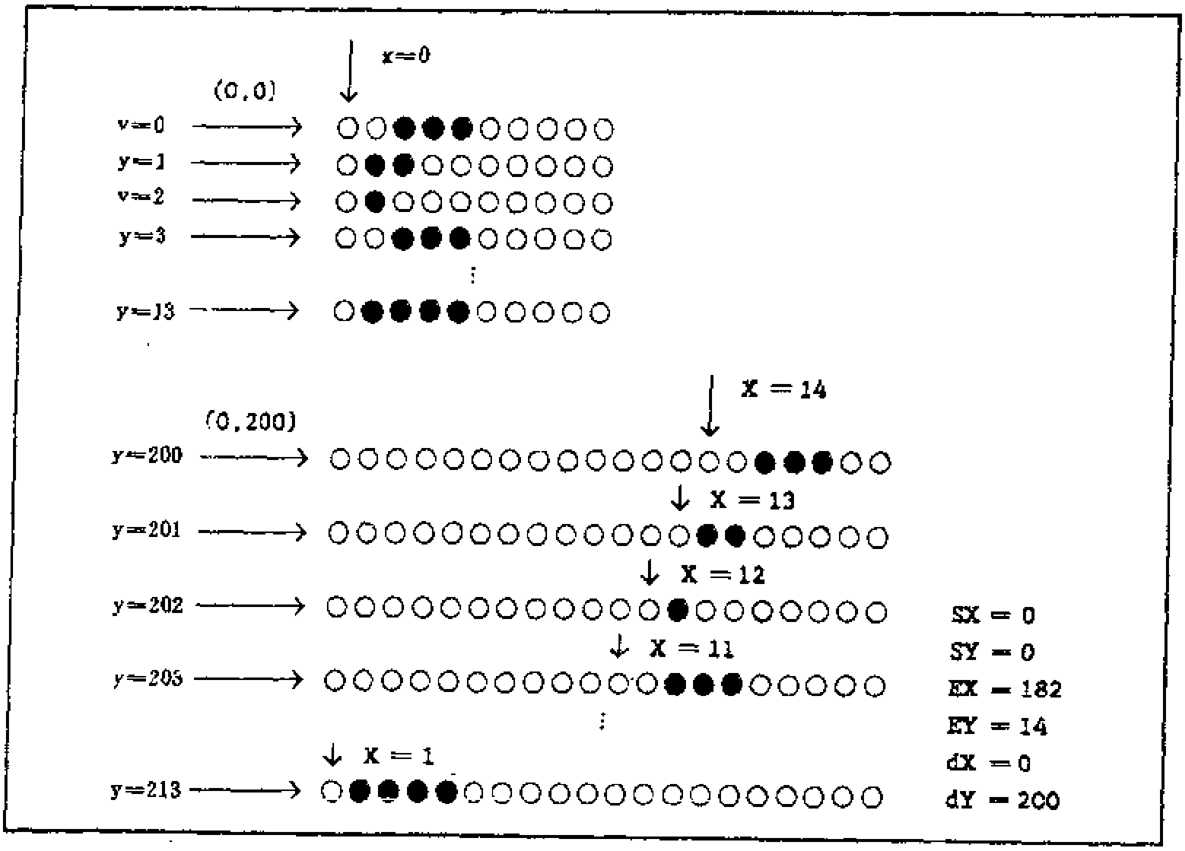


图 5-3 SLANT 函数的原理

这个程序的主要流程为：

1. 设定绘图模式为 640×350 256 色。
2. 在屏幕的左上角显示字串"Slanting character test"。
3. 将(0,0)-(182,14)区块的图形数据倾斜后，画在(0,200)的位置上。
4. 等候按键。
5. 设定为文本模式。

至于 Slant 函数的原理可以用图 5-3 来说明。

在这个程序中，我们以固定的斜率来变化设置区块内的图形数据。执行这个程序后，会在屏幕上面看到一排正体的英文字及一排斜体的英文字，当然你可以改写 Slant 函数，使它更具灵活性、更有实用价值。

5.3.4 画线函数

要在屏幕上画一条任意直线有许多种方法，其中最简单的莫过于利用直线方程 $y=mx+b$ 了。这个方程序中 m 及 b 都为常数，所以整个方程仅有 x 及 y 两个变量，我们可以选取 x 为自变量，y 则为因变量，利用一个循环连续改变 x 值即可算出连续的 y 值，一条直线便因此而画出。然而使用这种方法由于斜率$m\left(m=\dfrac{Y_2-Y_2}{X_2-X_1}\right)$是实数，为了顾及直线的准确度，整个式子必须使用实数运算，且在循环中每次都要执行一次实数乘法，使得这个方法效率低，直线绘制速度缓慢，因此大多数的场合都派不上用场，本文不用该法。

另一个画线的方法——Bresenham 画线法相比之下就效率高且速度快多了。假设我们要画一条直线如下图所示：

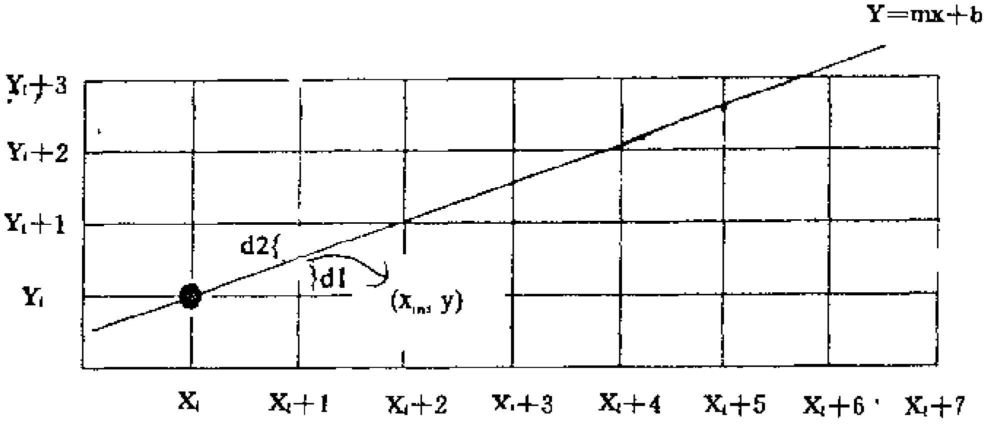


图 5-4 Bresenham 画线原理

如果在点(X_i，Y_i)处已经画好了一点，接下来要决定下一点究竟是要画在(X_i+1，Y_i)处还是(X_i+1，Y_i+1)处。这时我们就必须计算 Y 值的大小，然后比较 y 值是较接近 Y_i，还是较接近 Y_i+1。y 值的大小可依直线方程计算如下：

$$Y=m(X_i+1)+b$$

其中 m 为斜率，且 Y 值和 Y_i 的距离以 d1 表示为：

$d1 = Y - Y_i$

$= m(X_i+1)+b-Y_i$

而 Y 的值和 Y_i+1 的距离以 d2 表示为:

$d2 = (Y_i+1)-Y$

$= Y_i+1-m(X_i+1)-b$

接着再将两数 d1, d2 相减得:

$d1-d2 = (m(X_i+1)+b-Y_i)-(Y_i+1-m(X_i+1)-b)$

$= 2m(X_i+1)-2Y_i+2b-1$

因为$m = \dfrac{\triangle Y}{\triangle X}$, $\triangle X = X_2 - X_1$, $\triangle Y = Y_2 - Y_1$. 代入上式得到:

$$d1 - d1 = 2\frac{\triangle Y}{\triangle X}(X_i + 1) - 2Y_i + 2b - 1$$

两边各乘上△X 得到

$\triangle X(d1-d2) = 2\triangle Y(X_i+1)-2\triangle XY_i+2\triangle XY_i+2\triangle Xb-\triangle X$

$= 2\triangle XY_i+2\triangle Y-2\triangle XY_i = 2\triangle X(b-1)$

$= 2\triangle YX_i-2\triangle XY_i+c$

其中 C 为常数, $2\triangle Y+2\triangle X(b-1)$对每一点而言都相同 (因为同一条直线上有相同的△X 及△Y) 可以略去不提. 同时定义决定变量 $Di = \triangle X(d1-d2)$, 就可以确定:

如果 Di 为负数, 则 Y 比较接近 Y_i, 反之则较接近 Y_i+1.

上面这些推导过程可能稍嫌复杂, 如果读者想再深入研究, 可自行参阅计算图形学等相关书籍, 不然的话也可直接使用如下的流程图编写程序, 而无须完全了解该法的原理.

这个流程图是从 Bresenham 算法简化而来, 最重要的是进入画线的循环前, 先计算决定变量 d 的初值为:

$d = 2\triangle Y-\triangle X$

然后画上第一点(X_1,Y_1)后将 X 加 1, 判断是否小于零 (为负数), 如果是的话就不改变 Y 值, 使得下一点为(X_1+1,Y_1). 同时并改正决定变量的值为:

$d = d+2\triangle Y$

若 d 不小于零, 便把 Y 值加 1, 点画在(X_1+1,Y_1+1)处, 同时并改正决定变量的值为:

$d = d+2(\triangle Y-\triangle X)$

重复执行上述的画点, 判断并修正决定变量的步骤, 直到 X 到达 X2 为止, 一条直线便告完成。

使用这个方法画线时, 其中的决定变量 d 的初值只需在循环外计算一次即可, 而修正决定变量用的式子

$d = d+2\triangle Y$, 及

$d = d = 2(\triangle Y-\triangle X)$

中的 $2\triangle Y$ 及 $2(\triangle Y-\triangle X)$对每一点而言完全相同, 可在循环外先利用 $eq_1 = 2\triangle Y$ 及 $eq_2 = 2(\triangle Y-\triangle X)$计算一次后, 在循环中只须执行加法运算

$d = d+eq_1$ 及

$d = d+eq_2$ 就可以了.

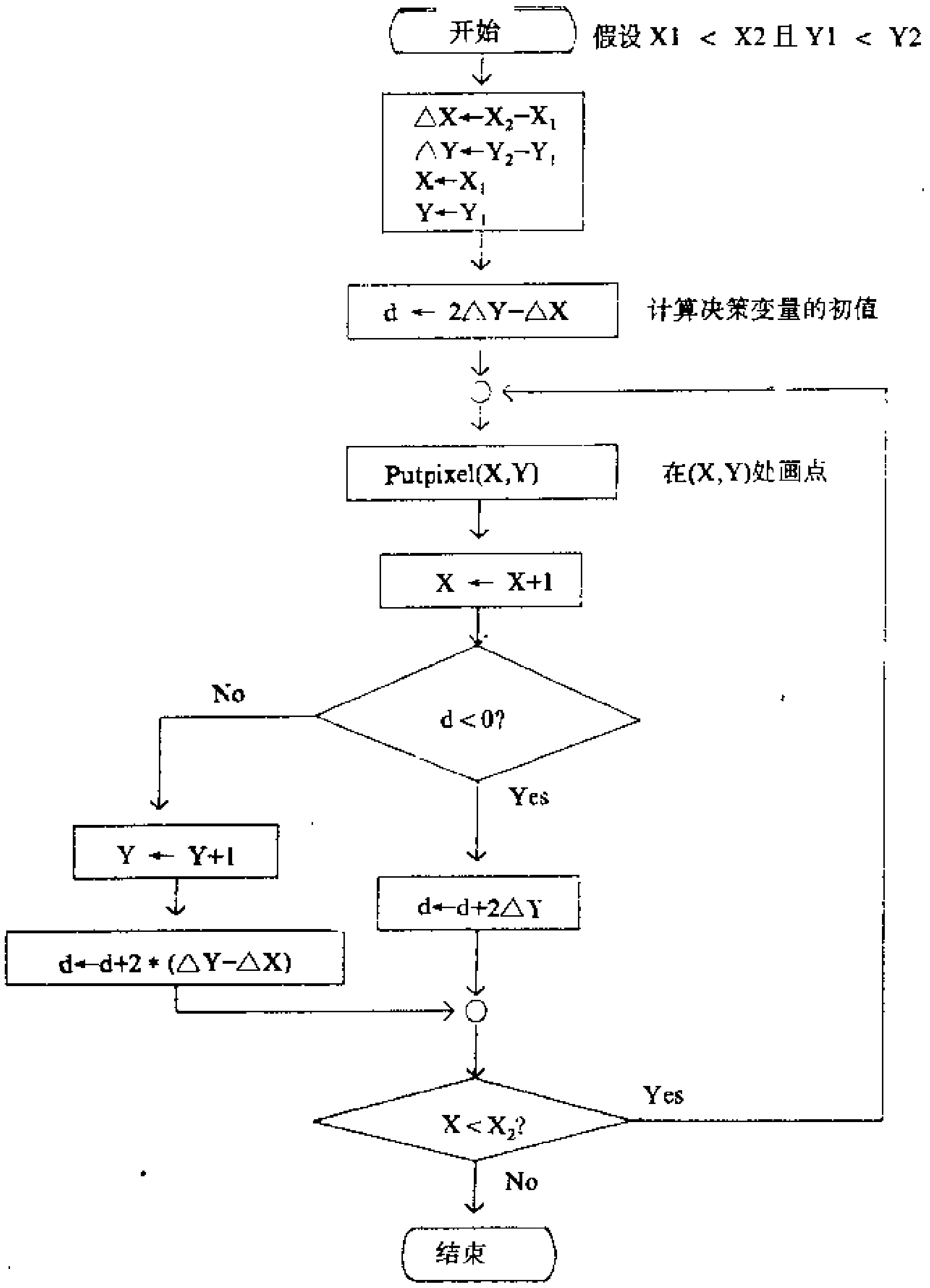


图 5−5　画线函数的流程图

由上可知，Bresenham 算法在画线的循环中完全不用乘法，且无须使用到实数，速度自然较其它的方法快而且效率高多了。接下来就是 LINE.C 程序及说明：

```
/* LINE.C */

#include "supervga.h"
#include "super.c"
#include <conio.h>
```

首先声明包含的文件名其中 Supervga.h 存放设定 Super VGA 的绘图模式的有关常数，而 Super.c 则放着画点常用程序 Putpixel 及设定绘图模式常数 Setvideomode。

```
void BLine(int, int, int, int, int);
```

函数原形定义如下：

```
main()
{
     Setvideomode( GC256__800x600 );
     BLine(0,0,799,599,14);
     getch();
     Setvideomode(TEXT__MODE);
}
```

画一条(0,0)−(799,599)颜色为黄色的直线，按任意键恢复到文本模式。

```
void BLine(int x1, int y1, int x2, int y2, int color)
{
   int deltax, deltay, x, y, eq1, eq2, d, incr;

   if(x1 > x2) {
     int temp;
     temp = x1;  x1 = x2;  x2 = temp;
     temp = y1;  y1 = y2;  y2 = temp;
   }
```

如果 X_1 大于 X_2，则交换(X_1,Y_1)及(X_2,Y_2)的内容，务必使 $X_1 < X_2$，令 X_1 为画线循环的起始点而 X_2 为终止点。

```
   if( y1 > y2 )
        incr = -1;
   else
        incr = 1;
```

如果 $Y_1 > Y3S = 2$，则 y 值设定为递减（增量为−1），反之则为递增。

```
   deltax = x2-x1;
   deltay = abs(y2-y1);
   d = 2 * deltay - deltay;
```

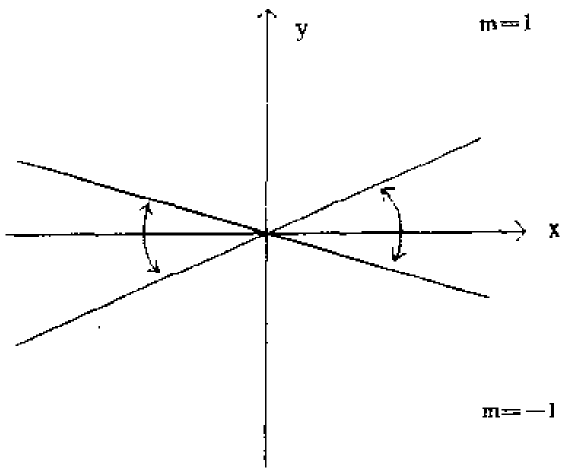
计算△X 及△Y 以及决定变量的初值。

```
   eq1 = 2 * deltav:
   eq2 = 2 * (deltay-deltax);
   x = x1;
```

计算修改决定变量用两个常数及设定 X，Y 之初值，从(X_1,Y_1)开始画。请注意：到目前为止所有的乘法都已在循环外做完，接下来的循环中只有加法的运算。

```
    do {
      Putpixel(x, y, color);
        x++;
        if(d<0)
              d += eq1;
        else  {
              y+=incr;
              d += eq2;
        }
    } while(x<=x2);
```

这个程序为了便于理解，把适用的情况限制在斜率为±1之间的直线，亦即如下图所示的范围内。



这段范围内的直线的△X大于等于△Y，所以上面的程序以X为自变量来画线。读者亦可使用Y为自变量编出斜率大于1或小于−1的画线程序，你不妨试试。Super VGA画线程序LINE.C的列表如下：

```
/* LINE.C */

#include "supervga.h"
#include "super.c"
#include <conio.h>

void BLine(int, int, int, int, int);

main()
{
     Setvideomode( GC256_800x600 );
     BLine(0,0,799,599,14);
     getch();
     Setvideomode(TEXT_MODE);
}
```

```
void BLine(int x1, int y1, int x2, int y2, int color)
{
    int deltax, deltay, x, y, eq1, eq2, d, incr;

    if(x1>x2) {
      int temp;
      temp=x1; x1=x2; x2=temp;
      temp=y1; y1=y2; y2=temp;
    }
    if( y1 > y2 )
          incr = -1;
    else
          incr =  1;
    deltax = x2-x1;
    deltay = abs(y2-y1);
    d = 2 * deltay - deltay;
    eq1 = 2 * deltay;
    eq2 = 2 * (deltay-deltax);
    x = x1;
    y = y1;
    do {
        Putpixel(x, y, color);
        x++;
        if(d<0)
              d += eq1;
        else  {
              y+=incr;
              d += eq2;
        }
    } while(x<=x2);
}
```

5.3.5 画圆函数

提到画圆，我们马上便会联想到圆的参数方程:

$x = r.\cos\theta$

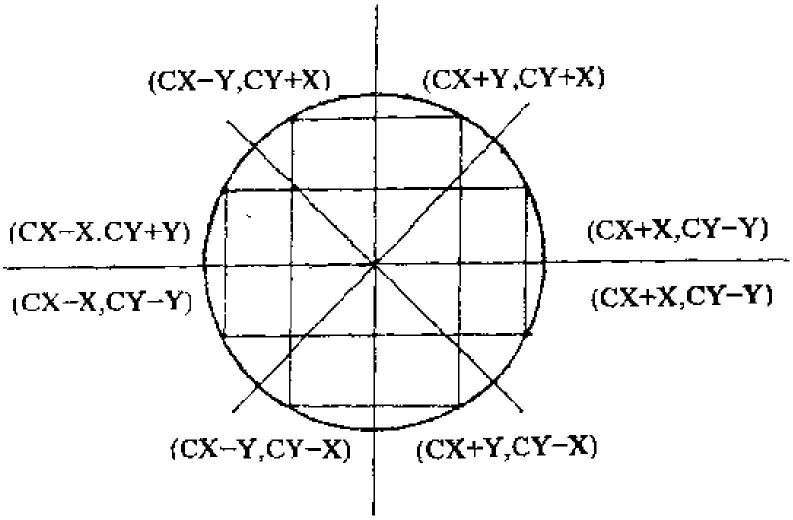
$y = r.\sin\theta$

其中 θ 为角度，而 r 为半径。利用这个式子，我们可以马上写一个画圆的函数来试试看:

```
/ *   FCIRCLE.C   * /
#include "supervga.h"
#include "super.c"
```

```c
#include <conio.h>
#include <math.h>

void FCircle(int, int, int, int);

main()
{
    Setvideomode(GC256_800x600);
    FCircle(400,300,200,14);
    getch();
    Setvideomode(TEXT_MODE);
}

void FCircle(int cx, int cy, int r, int color)
{
    void Put8cpixel(int, int, int, int, int);
    int x, y;
    float angle;

    for(angle=0; angle<0.7854; angle+=0.001) {
        x = (int) r * cos(angle);
        y = (int) r * sin(angle);
        Put8cpixel(x, y, color, cx, cy);
    }
}

void Put8cpixel(int x, int y, int color, int cx, int cy)
{
    Putpixel(cx+x, cy+y, color);
    Putpixel(cx-x, cy+y, color);
    Putpixel(cx+x, cy-y, color);
    Putpixel(cx-x, cy-y, color);
    Putpixel(cx+y, cy+x, color);
    Putpixel(cx-y, cy+x, color);
    Putpixel(cx+y, cy-x, color);
    Putpixel(cx-y, cy-x, color);
}
```

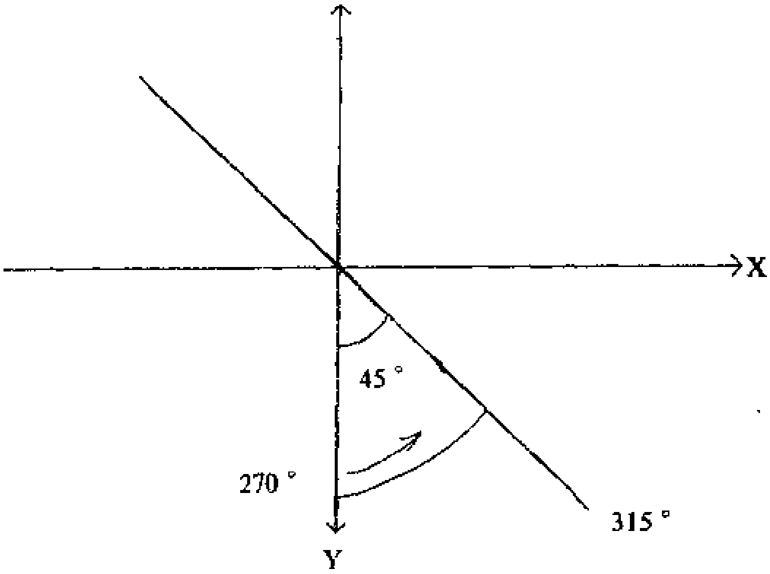
这个程序主要是将 angle（角度）代人参数求出 x 及 y 的值，再将点画在(x,y)处；由于圆的度数为 360，所以理论上必须将 0～360°各角度代人计算方能画出一圆。然而因为圆具有极高的对称性，所以我们只须计算 0～45°之间的值，而其它角度的点再利用对称性画出就好了（如图示）。



另外由于 Turbo C 的 Sin 及 Cos 函数都只接受弧度量来运算，45° $=\pi/4\approx$ 0.7854。而增量如果用 0.001 可以得到漂亮的圆形，然而却必须付出时间的代价。如果只要求执行速度快一点，而不在乎圆的质量，可以把增量改为 0.01 甚至 0.05 试试看。

概括来说，使用参数式来画圆由于使用到很多的实数乘法，它的速度依然不满足需求。所以在实用上，大部分的程序员仍然会选择 Bresenham 画圆算法。

Bresenham 画圆法基本上仍沿用着直线递增，并比较决定变量的方法，只是决策变量 D 的初值及其修改公式不同罢了。由于原理和直线法类似，在此就不多加说明，请读者自行阅读程序内容。值得一提的是，程序中所计算的弧的范围起自 270° 终至 315°，如下图所示：




```
/*  BFCIRCLE.C  */

#include "supervga.h"
```

```
#include "super.c"
#include <conio.h>
#include <math.h>

void FCircle(int, int, int, int);

main()
{
        Setvideomode(GC256_800x600);
        FCircle(400,300,200,14);
        getch();
        Setvideomode(TEXT_MODE);
}

void BCircle(int cx, int cy, int r, int color)
{
        void Put8cpixel(int, int, int, int, int);
        int x, y,d;

        x = 0;
        y = r;
        d = 3 - 2 *r;
        do    {
                Put8cpixel( x,y,color,cx,cy );
                if( d < 0 )
                        d = d + ( x << 2 ) + 6 ;
                else    {
                        d = d + ( (x-y) <<2 ) - 10 ;
                        y--;
                }
                x ++;
        } while( x<y );

        if( x == y )
                Put8cpixel(x,y,color,cx,cy);
}

void Put8cpixel(int x, int y, int color, int cx, int cy)
{
        Putpixel(cx+x, cy+y, color);
        Putpixel(cx-x, cy+y, color);
        Putpixel(cx+x, cy-y, color);
        Putpixel(cx-x, cy-y, color);
        Putpixel(cx+y, cy+x, color);
```

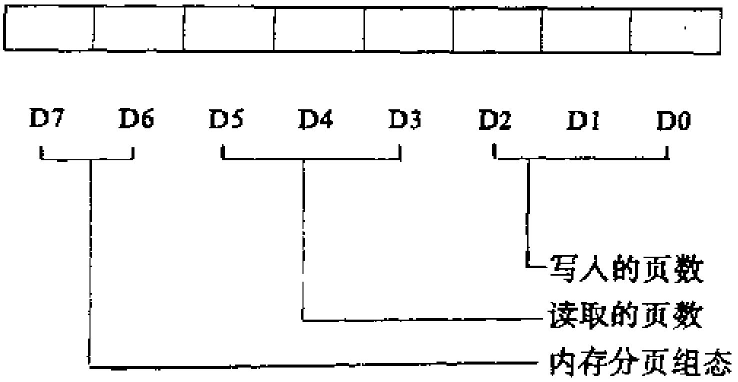
```
        Putpixel(cx-y, cy+x, color);
        Putpixel(cx+y, cy-x, color);
        Putpixel(cx-y, cy-x, color);
    }
```

5.4 直接存取 Super VGA 显示内存

5.4.1 ET3000 内存配置划分

要直接存取内存之前，首先必须了解 ET3000 对于 512K 的显示内存在各种扩充的绘图模式下是如何划分的．我们如何得知在各种模式下的内存配置呢？ET3000 提供了一个页选择寄存器（Page select register）供我们使用，它的 I／O 地址在 Port 3CDh，详细内容如下：

Port 3CDh



这个寄存器主要分成三部分，其中 D0～D2 允许程序设计者指定数据要写入显示内存的页数，而 D3～D5 则是指定数据要读出的页数．充分运用这两个部分(D0～D5)即可在 64KB 的可寻址范围内读取／写入到全部 512KB 的显示内存。

而寄存器第三部分(D6,D7)则报告目前的内存组织方式，详细情形如下：

表 5-3 Super VGA 显示内存分页配置

D7	D6	内存分页配置
0	0	可寻址范围 128K，分成四个位平面，共 512K
0	1	每页大小 64K，最多 8 页
1	0	512K 的连续内存，不分页也不分段
1	1	?

我们可以写一个程序报告各种绘图模式的配置，程序 SEGMENT.C 的列表如下：

```
/ *  SEGMENT.C  * /
#include <stdio.h>
#include <conio.h>
#include <string.h>
```

```c
#include <dos.h>
#include "supervga.h"
#include "super.c"

main()
{
    int code;
    char config[45];

    Setvideomode(GC256_800x600);
    code = inportb(0x3cd);
    Setvideomode(TEXT_MODE);
    code = code >> 6;
    switch(code) {
        case 0 : strcpy(config, "128KB addressable, 4 bit-plane");
            break;
        case 1 : strcpy(config, "64KB per page, 8 page maximum");
                break;
        case 2 : strcpy(config, "512KB of linear memory (no segments)");
                break;
        case 3 : strcpy(config, "Invalid");
                break;
}
    printf("Your Super VGA Extra Memory Paging Configurations are..\n");
    puts(config);
    puts("Press any key to continue...");
    getch();
}
```

这个程序执行后，会在屏幕的左上角显示：

```
Your Super VGA Extra Memory Paging Configurations are..
64KB per page,8 page maximum
Press any key to continue...
```

由此便可以知道在 800×600 256 色的模式下，显示内存是采用分页的技巧将内存分成每页 64KB(A000:0000～A000:FFFF)的大小，而全部 512KB 的内存最多可以分成 8 页。

事实上，如果执行过所有的扩充绘图模式后将会出现，其实只有 1024×768 16 色模式是使用 128K 的可寻址空间，而其它的扩充绘图模式全部都使用 64K 大小的分页技巧。在此将各种扩充绘图模式的内存配置整理如下，使读者能更容易理解并加以应用：

●800×600 16 色模式●

16 色模式所有的内存分成 4 个平面，所以本模式需要的寻址空间为(800／8)＊600／1024，约等于 58.59KB，在 A000:0000～A000:FFFF 的范围内，因此无

须使用到分页或 128KB 连续可寻址空间的技巧，它的内存全存取方式和标准 VGA 16 色绘图模式无异，其内存图解如下：

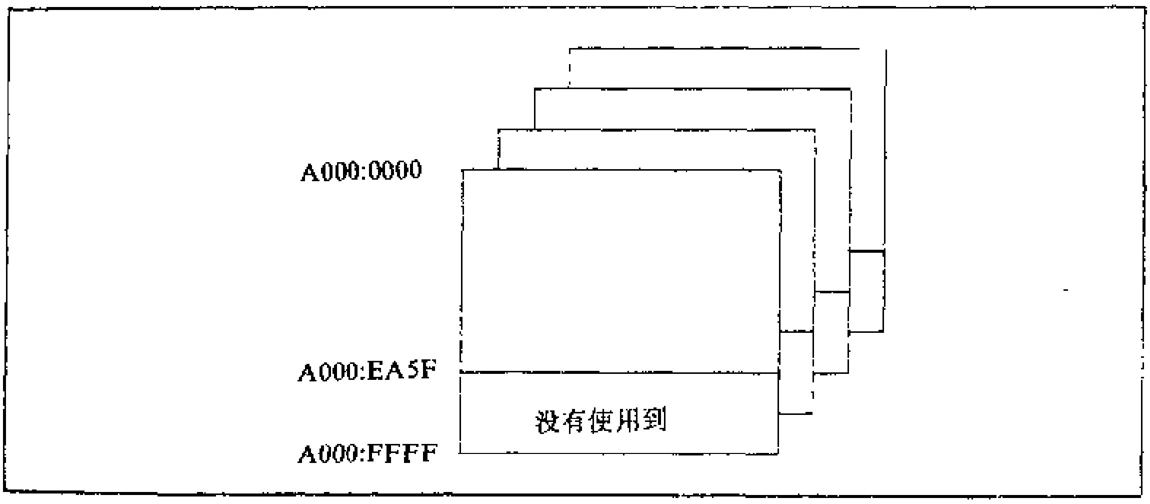


图 5-6　800×600 16 色模式的内存图解

●1024×768 16 色模式●

本模式所需寻址的空间为(1024／8)*768／1024 等于 96KB，超过了 A000:0000～A000:FFFF 的寻址范围，因此必须使用 128KB 的连续可寻址空间，也就是部分的图形数据必须存放于 B000:0000 以后的内存地址，其内存图解如下：

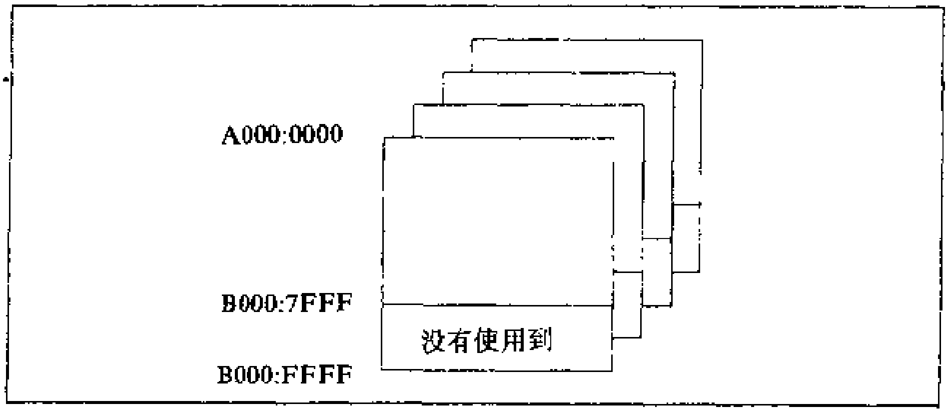


图 5-7　1024×768 16 色模式的内存图解

●640×350 256 色模式●

256 色模式的内存架构和 16 色模式完全不同。在这种模式下使用 1 个 byte 来表示一个像素点(pixel)，所以所需的内存更多，只有使用分页的技巧才能完成。

这个模式所需要的内存为 640×350 等于 218.75KB，因为一页的大小为 64KB，所以显示内存分成 218.75／64≈3.4 至少 4 页，图解如下：

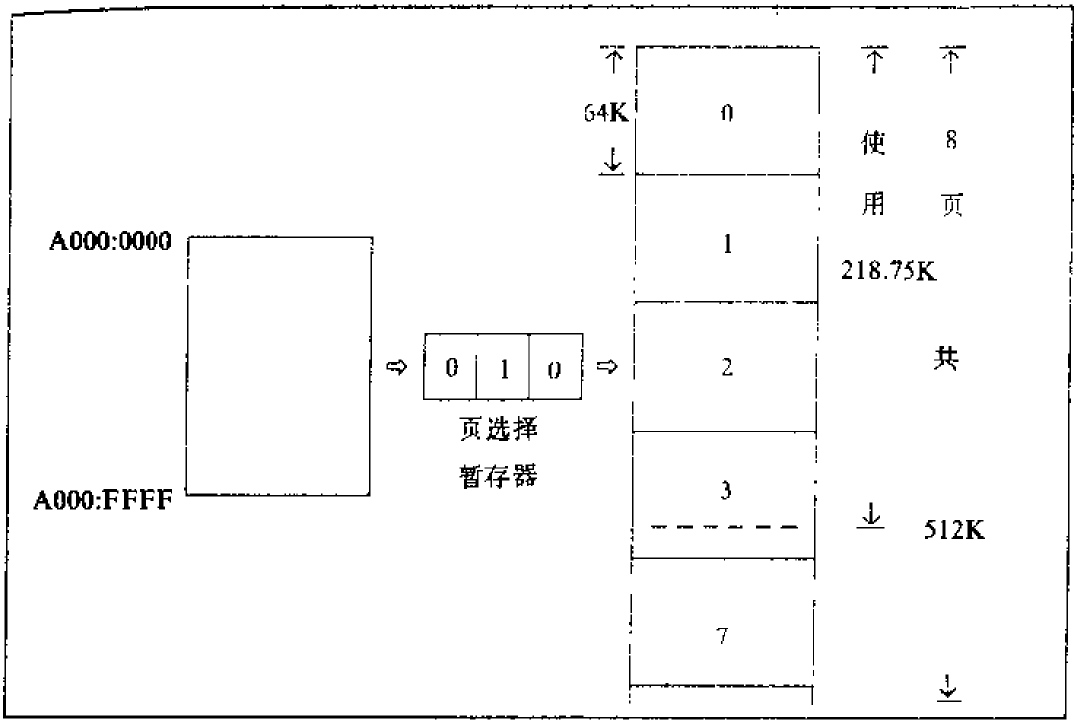


图 5-8　640×350 256 色模式的内存图解

●640×480 256 色模式●

和上一个模式类似，只是所使用到的内存为 640×480＝300KB，显示内存分成 5 页，图解如下：

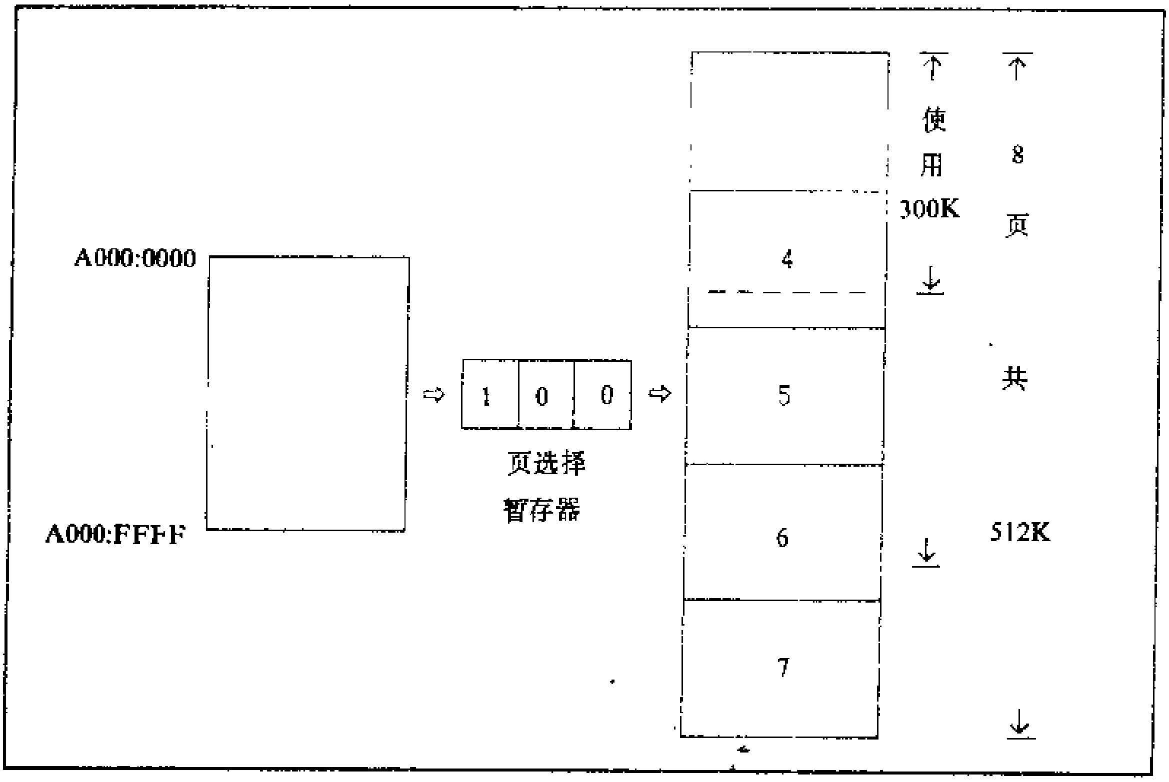


图 5-9　640×480 256 色模式的内存图解

●800×600 256 色模式●

这个模式所占内存 800×600=468.75KB，使用到全部的页数（8 页），图解如下：

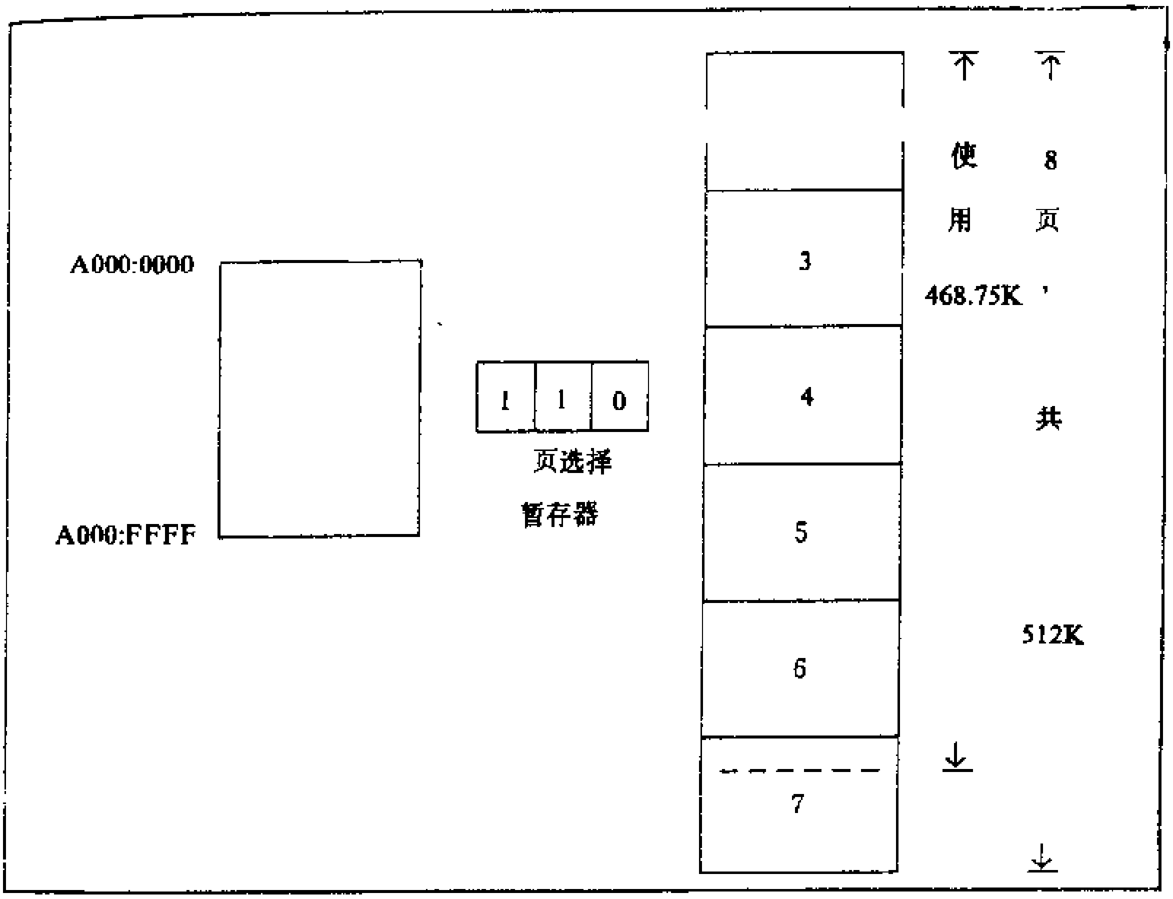


图 5-10 800×600 256 色模式的内存图解

5.4.2 将图形数据直接写入显示内存

有了上一个小节所介绍的概念，现在就可以写一个屏幕填色的程序验证看看，请看 FILL1.C：

```
#include <conio.h>
#include "supervga.h"
#include "super.c"

main()
{
    long i;

    Setvideomode(GC256_800x600);
    for(i=0; i<65536; i++)
        pokeb(0xa000, i, 33);
    getch();
    Setvideomode(TEXT_MODE);
}
```

这个程序主要是将64KB的数据33（蓝色）存入A000:0000～A000:FFFF的范围内，由于并没有指定页数，适配器预设是第0页，即是在屏幕的上方涂上约屏幕八分之一大小的蓝色长方形（因为800×600 256色模式的全页图形数据共分8页储存，这个程序只存入其中的一页，故占八分之一），如果要填满全屏幕，那么就必须利用页选择寄存器来改变读／写的页数了。

5.4.1节曾经提过页寄存器的I／O地址在3CDh处，其中D0～D2指定写入的页数，而D3～D5设定读取的页数。在此就利用这个寄存器编写两个有用的函数Select_wpage及Select_rpage如下:

```
void Select_wpage(int page)
{
   char code ;
   if((page>8)||(page<0))
          return;
```

先检查页数(Page)的范围有没有超过最大的页数，如果是负数也不能接受。

```
code = inportb( 0x3cd);
```

利用inportb函数先取得页选择寄存器原来的值，放在变量code中。

```
code & = 0xf8;
```

这段程序码是将原来页选择寄存器的内容保存，但是用逻辑运算AND将D0～D2的3个bit清除为0。假设原来的code的值为01001011b，运算过程如下:

```
          原来code的值=01001011
                 0xf8 = 11111000
                 AND)
          ─────────────────────
经过AND运算的code的值  =01001000
```

```
code | = page;
```

将D0～D2清除为0后，接着就把page的值利用逻辑运算OF写入D0～D2的位置。假设要写入第4页，则运算过程如下:

```
经过AND运算的code值=01001000
      page内容     =0000010
             OR)
  ─────────────────────
      新的code值   =01001100
```

```
   outportb(0x3cd,code);
}
```

最后再将经过运算得到的新值利用outportb函数写入页选择寄存器，即完成对写入页数的设定工作。读取页数的设定原理也相同，不再多加说明，仅列表于后，并将这两个函数加入SUPER.C中以供日后使用。

```
void Select_rpage (int page)
```

```
{
    char code;

    if( (page>8 ) || (page<0) )
        return;
    code =inportb( 0x3cd );
    code &=0xc7;
    code |= ( page<<3);
    outportb( 0x3cd,code);
}
```

现在我们就可以使用 Select__wpage 函数将整个屏幕填满某种颜色了。(FILL2.C)

```
#include <conio.h>
#include "supervga.h"
#include "super.c"

main()
{
    long i;
    int  page;

    Setvideomode(GC256__800x600);
    for(page=0; page<8; page++) {
        Select__wpage(page);
        for(i=0; i<65536; i++)
          pokeb(0xa000, i, 33);
    }
    getch();
    Setvideomode(TEXT__MODE);
}
```

截至目前为止，所使用的分页方法并不适合 1024×768 16 色模式，如想在 1024×768 16 色模式下填满整个屏幕，其方法如下所示：

```
#include <conio.h>
#include "supervga.h"
#include "super.c"

main(
{
    long i;

    Setvideomode( GC16__1024x768 );
    outportb( 0x3c4,2 );
    outportb( 0x3c5,9 );
```

```
    for( i=0;i<65536;i++ );
        pokeb( 0xa000,i,0xff );
    for( i=0;i<32767;i++ );
        pokeb( 0xb000;i,0xff );
    getch();
    Setvideomode( TEXT_MODE );
}
```

因为 16 色模式并不是 8 个 bits 代表一个像素点，所以使用 pokeb 所存入的值 0xff 并不代表某一像素点的颜色值，而是指该邻近的 8 个像素点全部点亮(turn on)，至于所点亮的是代表该 byte 地址所对应到的 4 个平面(Plane)的哪些平面，则是看平面写入使能寄存器的值为多少而定，所以在标准 VGA 所使用的：利用平面写入寄存器来控制像素点颜色的方法在 Super VGA 下仍然有效，所以这个程序利用 outportb (0x3c4,2)；outportb (0x3c5,9)；将屏幕上存入的颜色设定为天蓝色，如果拿掉这两个程序，屏幕变成白色(预设值是 4 个平面都使能)。

另外别忘了，1024×768 16 色模式的内存划分是在 A000:0000～B000:7FFF 之间的一块连续可寻址空间，所以要存入数据并无需使用页选择寄存器，直接将数据存入该地址即可。

5.4.3 直接绘点法

●256 色模式绘点法●

在 Super VGA 256 色模式下绘点必须注意到三个参数，分别为段地址 Segment、偏移地址 offset 及页数 page。其公式如下：

Segment：恒为 A000h

offset=(width * Y+X)／10000h

page=(width * Y+X)%1000h

其中 width 为每条扫描线的 byte 数，而(x,y)就是要画的像素点的逻辑坐标。使用这个公式要写画点的程序就不难了，只要上面这段公式换成程序码，利用得到的参数将数据存入显示内存即可，下面这段程序就是一个完整的 256 色绘点函数。

```
void Writepixel256(int x, uint y, int color, int width)
{
    unsigned offset, page;

    page   = ( (long) width * y+x ) / 0x10000;
    offset = ( (long) width * y+x ) % 0x10000;

    Select_wpage( page );
    pokeb( 0xa000, Offset, color );
}
```

这个函数的写法很直接，然而它的执行效率却比调用 BIOS 的写点功能还差，执行速

度还要慢．为什么呢？因为它使用到了一个长整数的除法及一个长整数的取余数，这两个运算是执行速度快慢的最大关键，所以我们将它改写如下：

```
void Writepixel256( int x, uint y, int color, int width)
{
    unsigned offset, page;

    page    = ( (long) width * y+x ) >>16;
    offset  = ( (long) width * y+x ) & 0xffff;

    Select_wpage( page);
    pokeb( 0xa000, offset, color );
}
```

将其中的除法和余数运算用位移及 AND 运算取代，所增加的执行速度就相当可观了．究竟相差多少呢？根据笔者试验的结果，执行速度几乎快上 2 倍．

使用我们自己的方法的画点程序究竟和原来的使用 BIOS 功能调用的 Putpixel 函数的执行速度相差多少呢？我们就分别写两个程序来计算看看．第一个程序为使用 Putpixel 的例子：

```
/ * WPIXBIOS.C * /

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#include <time.h>
#include <bios.h>
#include "supervga.h"
#include "super.c"

main()
{
    int x, y, color;
    long i;
    long tstart, tend;
    float tused;

    randomize();
    Setvideomode(GC16_1024x768);
    tstart = biostime(0, 0);
    for(i=0; i<50000; i++) {
        x = random(1024);
        y = random(768);
        color = random(16);
```

```
        Putpixel(x, y, color);
    }
    tend = biostime(0, 0);
    Setvideomode(TEXT_MODE);
    tused = (unsigned float) (tend-tstart) / 18.204;
    printf("Draw 50000 pixeles using BIOS function
           call need %2.4f seconds\n", tused);
    getch();
}
```

这个程序在执行后，会在屏幕上先用随机函数画上 50000 个像素点，然后在屏幕上方出现如下字样：

DRAW 5000 pixeles using BIOS function call need 3.9552 seconds

表示在这个模式下，随机画 50000 个像素点需要 3.9552 秒的执行时间（产生随机数的时间约为 0.9888 秒），当然这个数据随所使用的计算机不同而有所不同（笔者所使用的 80386-40 的 PC，时钟频率为 65MHz），所以这个数字单独看并没有什么意义，但是在同一台计算机上执行另一个函数（当然是指目的相同的函数）就可以比较出两种方法的优劣了。

接下来你一定很有兴趣地想知道我们所写的这个 Writepixel256 的速度如何，先来执行下面这个程序：

```
/* FWPIX256.C */

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#include <time.h>
#include <bios.h>
#include "supervga.h"
#include "super.c"

void Writepixel256(int, int, int, int);

main()
{
    int x, y, color;
    long i;
    long tstart, tend;
    float tused;

    randomize();
    Setvideomode(GC256_800x600);
```

```
    tstart = biostime(0, 0);
    for(i=0; i<50000; i++) {
          x = random(800);
          y = random(600);
          color = random(256);
          Writepixel256(x, y, color, 800);
    }
    tend = biostime(0, υ);
    Setvideomode(TEXT_MODE);
    tused = (unsigned float) (tend-tstart)/18.204;
    printf(" Draw 50000 pixeles using Direct-Access method
              need %2.4f seconds\n", tused);
      getch();
}

void Writepixel256(int x, int y, int color, int width)
{
      unsigned offset;
      unsigned page;

      page = ((long) width * y + x) >> 16;
      offset = ((long) width * y + x) & 0xffff;

      Select_wpage(page);
      pokeb(0xa000, offset, color);
}
```

这个程序执行后显示:

Draw 50000 pixeles using Direct-Access method need 1.8182 seconds

速度刚好是使用 BIOS 功能调用的一倍，这样的画点函数实用价值就很高了.

●1024×768 16 色模式绘点法●

在 16 色绘图模式中，只有这个模式较为特殊，需要特别介绍，至于其它的 16 色模式则只需沿用原来标准 VGA 的绘点法即可。

这个模式由于可寻址的范围由 A000:0000 到 B000:7FFF 为止，超过一个段，因此我们把它分成两个部分来处理:

第一部分: A000:0000～A000:FFFF

第二部分: B000:0000～B000:7FFF.

公式归纳如下:

addr=width * Y+X/8

如果 addr 大于 65536 则

Segmetn=B000h

addr=addr-65536

否则 Segment=A000h

bitposition=X%8

其中 bitposition 为像素点在该 byte 中的位置。请记住：在 16 色模式中一个字节代表 8 个像素点，而各点的颜色则由 4 个位平面来决定。因为每个字节代表 8 个像素点，所以在写入一个像素点时，为了不影响到相邻的像素点，必须先把原来的那个地址上的图形数据保存起来，用 OR 运算将该像素点的值写入，然后再存入该字节。但是在存入字节之前，请记得更改位屏蔽寄存器(Bit Mask Register ,3CEh index 8)的值，以免影响到其它相邻的像素点。同时亦可以使用比特平面屏蔽寄存器(Plane Mask Register ,3c4,index 2)来决定写入像素点的颜色。函数 Writepixel16 的列表如下：

```
void Writepixel116(int x, int y, int color, int width)
{
    unsigned segment;
    long addr;
    int bitposition, shift;
    char origin;

    addr = ( (long) width  * y+ (x> >3) );
    if(addr>65536)  {
       segment = 0xb000;
       addr -= 65536;
    }
    else
       segment = 0xa000;
       bitposition = 7 - x & 7;
       origin = pokkeb( segment, addr );
       shift = 1 < < bitposition;
       origin |= shift;
       outportb(0x3ce, 8);
       outportb(0x3cf, shift);
       outportb(0x3c4, 2);
       outportb(0x3c5, color);
       pokeb( segment, addr, origin );
}
```

这个函数的执行速度有多快呢？我们仍旧写一个程序来测量看看：

```
/ * FWPIX16.C * /

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#include <time.h>
```

```
#include <bios.h>
#include "supervga.h"
#include "super.c"

void Writepixel16(int, int, int, int);

main()
{
    int x, y, color;
    long i;
    long tstart, tend;
    float tused;

    randomize();
    Setvideomode(GC16_1024x768);
    tstart = biostime(0, 0);
    for(i=0; i<50000; i++) {
        x = random(1024);
        y = random(768);
        color = random(16);
        Writepixel16(x, y, color, 128);
    }
    tend = biostime(0, 0);
    Setvideomode(TEXT_MODE);
    tused = (unsigned float) (tend-tstart) / 18.204;
    printf("Draw 50000 pixeles using Direct-Access method
            need %2.4f seconds\n", tused);
    getch();
}

void Writepixel16(int x, int y, int color, int width)
{
    unsigned segment;
    long addr;
    int  bitposition, shift;
    char origin;

    addr = ( (long) width * y + (x>>3) );
    if(addr>65536) {
        segment = 0xb000;
        addr -= 65536;
    }
    else
        segment = 0xa000;
```

```
        bitposition = 7 - x & 7;
        origin = peekb( segment, addr );
        shift = 1 << bitposition;
        origin |= shift;
        outportb(0x3ce, 8);
        outportb(0x3cf, shift);
        outportb(0x3c4, 2);
        outportb(0x3c5, color);
        pokeb( segment, addr, origin );
    }
```

执行的结果为:

Draw 50000 pixeles using Direct-Access method need 1.8677 seconds

那么这个模式下使用 BIOS 调用的写点函数执行速度怎样呢？结果如下:

Draw 50000 pixeles using BIOS function call need 5.7680 seconds

显然在 1024×768 16 模式下使用自己写的函数所得到的效率更高，速度几乎快了两倍.

以上这两个模式下的写点函数，应用在 Super VGA 下的一般绘图应该是已经足够了；但是如果要应用在动画上的话，恐怕要使用查表法改写或是直接以汇编语言来完成了.

800×600 256 色及 1024×768 16 色两种模式下的查表绘点法如下所示，至于汇编语言就看读者自己发挥了.

```
/* TWPIX256.C */

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#include <time.h>
#include <bios.h>
#include "d:\book\ch5\supervga.h"
#include "d:\book\ch5\super.c"

void Writepixel256(int, int, int);
void tabulate(void);

long table[600];

main()
{
    int x, y, color;
    long i;
```

```
    long tstart, tend;
    float tused;

    randomize();
    tabulate();
    Setvideomode(GC256_800x600);
    tstart = biostime(0, 0);
    for(i=0; i<50000; i++) {
          x = random(800);
          y = random(600);
          color = random(256);
          Writepixel256(x, y, color);
    }
    tend = biostime(0, 0);
    Setvideomode(TEXT_MODE);
    tused = (unsigned float) (tend-tstart) / 18.204;
    printf(" Draw 50000 pixeles using Direct-Access method need %2.4f seconds\n", tused);
    getch();
}

void Writepixel256(int x, int y, int color)
{
     unsigned offset;
     unsigned page;

     page = (table[y] + x) >> 16;
     offset = (table[y] + x) & 0xffff;

     Select_wpage(page);
     pokeb(0xa000, offset, color);
}

void tabulate()
{
     int i;

     for(i=0; i<600; i++)
          table[i] = (long) 800 * i;
}
```

以上程序在 800×600 256 色下的查表法绘点的速度为:

Draw 50000 pixeles using Direct-Access method need 1.5381 seconds

请记住原来随机数的产生就费时 0.9888 秒了，所以使用查表法给 50000 个像素点仅仅只要 0.5493 秒，比原来未使用查表法时的 0.824 秒又改进了不少。

接着是在 1024×768 16 色模式下查表法程序 TWPIX16.C 的列表:

```
/ * TWPIX16.C * /

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#include <time.h>
#include <bios.h>
#include "supervga.h"
#include "super.c"

void Writepixel16(int, int, int);
void tabulate(void);

long table[768];

main()
{
    int x, y, color;
    long i;
    long tstart, tend;
    float tused;

    randomize();
    tabulate();
    Setvideomode(GC16_1024x768);
    tstart = biostime(0, 0);
    for(i=0; i<50000; i++) {
         x = random(1024);
         y = random(768);
         color = random(16);
         Writepixel16(x, y, color);
    }
    tend = biostime(0, 0);
    Setvideomode(TEXT_MODE);
    tused = (unsigned float) (tend-tstart)/18.204;
    printf(" Draw 50000 pixels using Direct-Access metnod
            need %2.4f seconds\n", tused);
    getch();
}

void Writepixel16(int x, int y, int color)
```

```
{
    unsigned segment;
    long addr;
    int  bitposition, shift;
    char origin;

    addr = (table[y]+ (x>>3));
    if(addr>65536) {
        segment = 0xb000;
        addr -= 65536;
    }
    else
        segment = 0xa000;
    bitposition = 7 - x & 7;
    origin = peekb(segment, addr);
    shift = 1 << bitposition;
    origin |= shift;
    outportb(0x3ce, 8);
    outportb(0x3cf, shift);
    outportb(0x3c4, 2);
    outportb(0x3c5, color);
    pokeb(segment, addr, origin);
}

void tabulate()
{
    int i;

   for(i=0; i<768; i++)
      table[i] = (long) 128 * i;
}
```

这个程序的执行结果为:

Draw 50000 pixeles using Direct-Access method need 1.7579 seconds

也改进了不少，最后在此就将各方法的效率列表整理，做为本节的结束。

表 5-4 各种绘点法的效率比较

使用机器: 80386DX-40	65MHz
测试方法: 随机数绘	50000点
随机数循环共占	0.988秒
256 色模式	
BIOS调用法	3.9552秒

直接绘点法	1.8128秒
查表绘点法	1.5381秒
1024X768 16 色模式	
BIOS调用法	5.7680秒
直接绘点法	1.8677秒
查表绘点法	1.7579秒

5.5 硬件图形放大功能

5.5.1 与画面放大有关寄存器

ET-3000 卡提供了硬件的画面放大功能，现在就让我们来看看如何去使用它。

首先，让我们先看看在表 5-5 中，和画面放大有关的各 Super VGA 寄存器的说明：

表 5-5 与画面放大有关的寄存器

I/O 地址	寄存器名称	说明
3C5 Index 6	Zoom Control	设定放大功能及放大倍数
3D5 Index 1Bh	X Zoom Start	设定放大后区域的 X 坐标
3D5 Index 1Ch	X Zoom End	
3D5 Index 1Dh	Y Zoom Start	设定放大后区域的 Y 坐标
3D5 Index 1Eh	Y Zoom End	
3D5 Index 1Fh	Y Start/End High	
3D5 Index 20h	Zoom Start Low	被放大的区域距离放大区
3D5 Index 21h	Zoom Start High	域起始地址的偏移值
3D5 Index 23h	Extended Start Address	

基本上，要在屏幕上放大某一区块，必须先告诉 ET3000 所要放大的区域在哪里、以及放大的倍数多大等参数。这些参数就只有通过上表所列的寄存器才能与之沟通，而沟通的方式就如标准 VGA 一般，使用 outportb 或 outport 函数，指定 I/O 的地址及下标值，便可将数据写入该寄存器了。假设我们要写入一值 35 到 3D5h，Index 1Bh 这个寄存器，那么可以使用以下的方法：

```
outportb(0x3d4,0x1b);
outportb(0x3d5,35);
```

其中，3d4h 就是代表这组寄存器的下标部分，而 3d5h 就是数据部分。以上的方法也可以用下面这种方法来取代：

```
code=0x1b;
code!=35<<8;
outport(0x3d4,code);
```

这样只需执行一次 I/O 的写入动作，outport 函数便会自动的将 code 的低地址部分

(1Bh)写入 3D4h，而将高地址部分(35)写入 3D5h。本节将采用后面这种方式。

接着请看画面放大及有关寄存器的示意图：

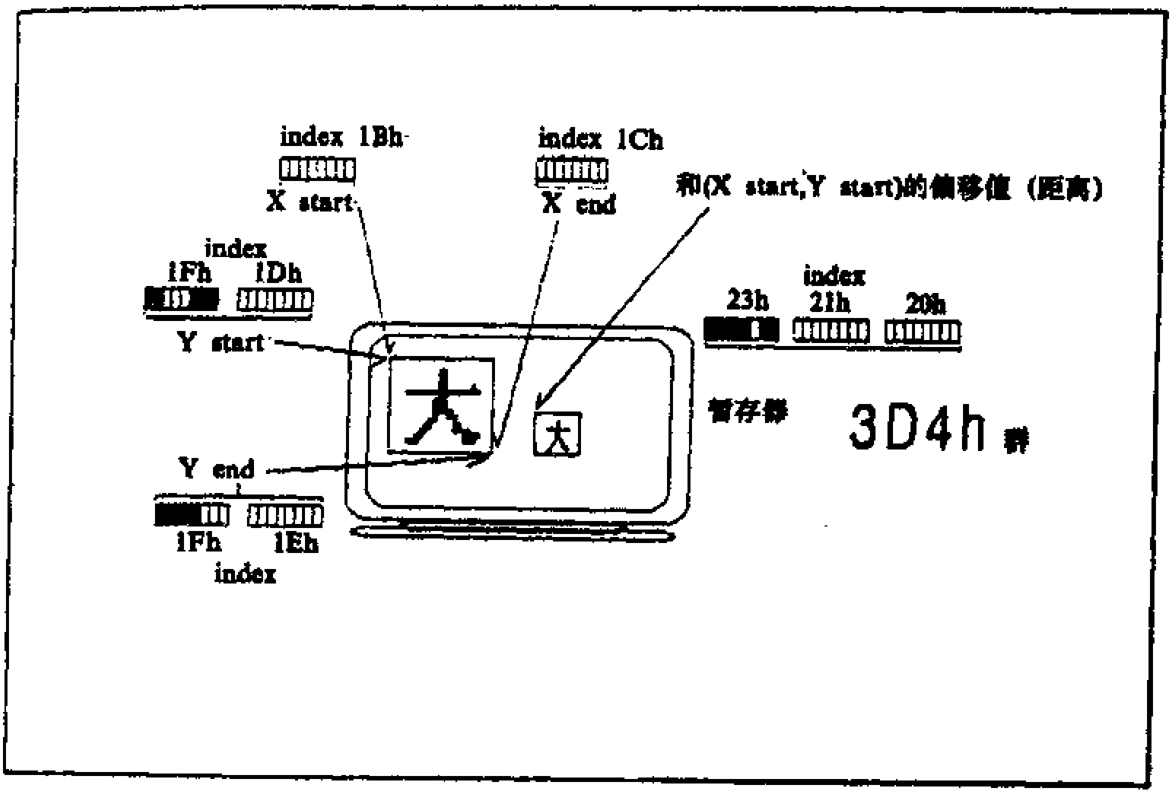


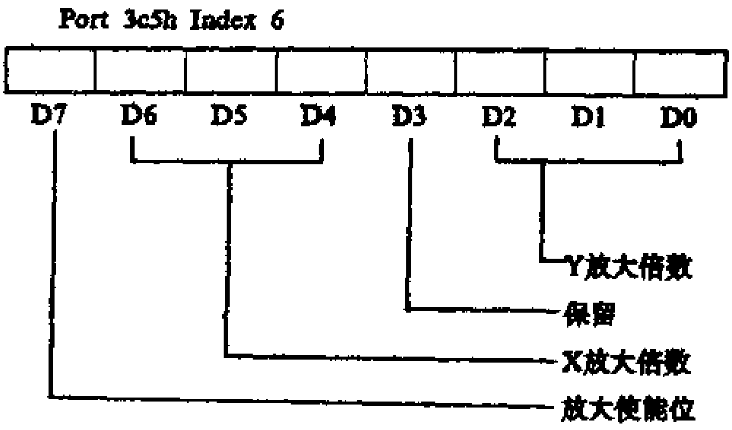
图 5–11 画面放大及相关寄存器

5.5.2 画面放大的步骤

画面放大的步骤如下：

1. 设定放大功能及放大的倍数

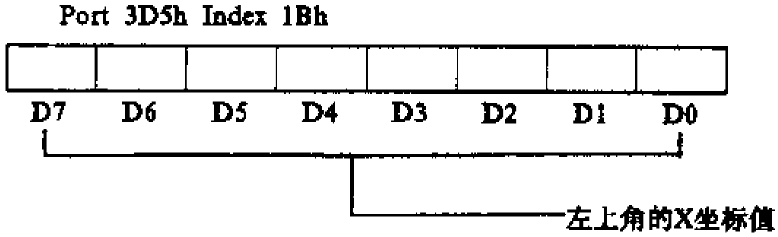
这项工作可通过寄存器 3c5h，Index 6 完成，这个寄存器的全部 8 个位由放大使能、X 放大倍数及 Y 放大倍数三部分所构成，其位分解图如下所示：



由于指定放大倍数的部分(D0～D2 及 D4～D6)只有 3 个位，所以放大倍数的范围只有 0～7 八种倍数。如果要放大画面，放大使能位(D7)一定要设定为 1。

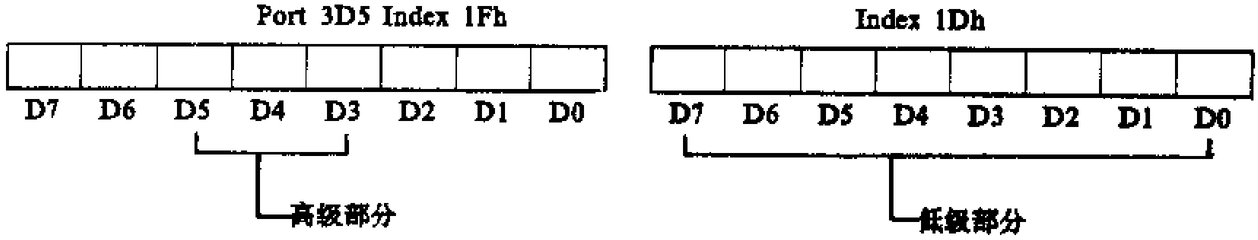
2. 设定放大后区块的左上角 X 坐标值

设定 X-START 的值可以由寄存器 3D5,Index 1Bh 完成，由于本寄存器中的值是以字符宽度为单位，通常以 8 个像素点来计值，所以不论是 16 色模式或 256 色模式，一个字节就足够容纳这些数值了。



3. 设定放大后区块的左上角 Y 坐标值

由于 Y 坐标值是以扫描线为计数单位，而最大的扫描线模式总共有 768 条，超出一个位所能表示的数值，必须使用两个寄存器，分别是 3D4h,Index 1Dh 及 Index 1Fh。其中 1Dh 使用到全部的位，而 1Fh 只使用到其中的 3 位，加起来共 11 位，最多可以表示到 $2^{11}=2048$，足够应付现在的使用及未来的扩充。



4. 设定放大后区块的右下角 X 坐标及 Y 坐标值

如同 2 和 3 所解释，右下角的坐标值设定方法和左上角一致。然而，大部分在放大时所指定的参数只有放大后的左上角坐标，放大倍数及被放大的区块位置大小等，因此放大后区块的右下角坐标便要使用这些参数来计算才有设定的依据。其公式如下：

XEND = (XSCALE+1) * Window__Width+XSTART

YEND = (YSCALE+1) * Window__Height * Char__Height+YSTART

XSCALE = X 的放大倍数

YSCALE = Y 的放大倍数

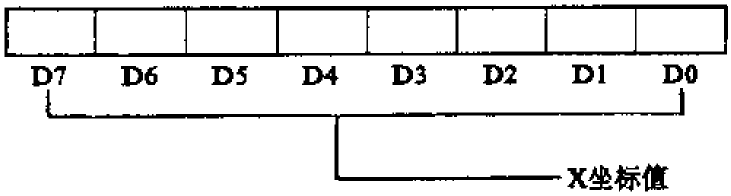
Window__Width：要放大窗口的宽度，以字符宽度为单位。

Window__Height：要放大窗口的高度。

Char__Height：字符的高度。

设定 X 坐标的寄存器如下：

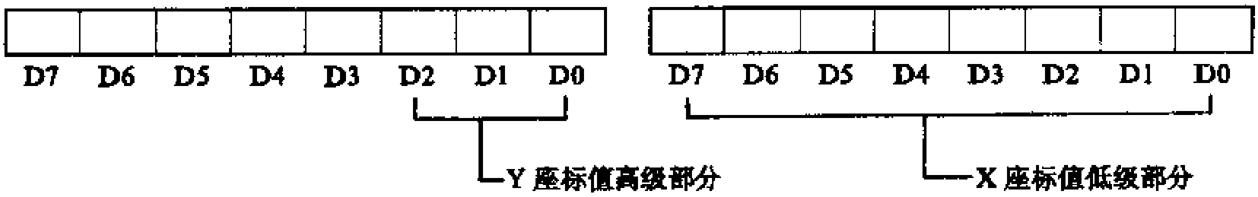
Port 3D5 Index 1Ch



而设定 Y 坐标值依然由两个寄存器组成

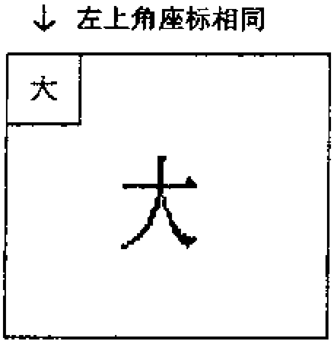
Port 3D5 Index 1Fh

Index 1Dh



5. 设定要放大区块左上角地址和放大后区块左上角坐标的偏移值

由于这个偏移值是两个坐标的实际内存地址相减后得到的值，可能会很大，所以由三个寄存器来组成，但是大部分的情形都是两个坐标重叠，如下图所示：

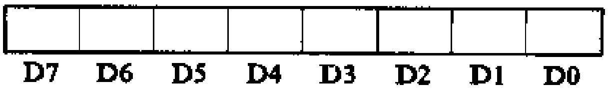


所以这些寄存器的内容存入 0 即可。寄存器分解如下：

Port 3D5h Index 23h

Index 2h

Index 20h



只要照着以上的步骤一步步完成，便可顺利地写出画面放大的程序，接下来我们就写一个程序加以验证：(ZOOM.X)

```
#include <dos.h>
#include <stdio.h>
#include <bios.h>
#include <time.h>
#include <stdlib.h>
#include "supervga.h"
#include "super.c"
#include "key.h"

int Screen_Width;
int Screen_Height;
int Char_Height;
int Screen_Rows;
int Window_Width =2;
int Window_Height =2;
```

Screen_Width 是屏幕的宽度，以 byte 为单位，Screen_Height 是屏幕的高度，以扫描线为单位，而 Char_Height 则是该模式下字符的高度，Screen_Rows 是列数. 因为图形放大的功能并不局限在某种绘图模式下，所以必须通过这些变量才能计算放大所需相关参数.

Window_Width 及 Window_Height 则分别记录要放大的区域宽高，以绘图文字的大小为单位.

```
void Writepixel256(int, int, int);
void BCircle(int, int, int);
void tabulate(void);
void Setvideomode(int);
void Zoom(int, int, int, int);
void Readinfo(void);
void Action(void);
void Draw_Screen(void);

long table[600];
```

定义函数型. 长整型数组则是用来存放扫描线的起始地址，以供绘点函数 Write_Pixle256 使用.

```
main()
{
    Setvideomode(GC256_800x600);
    tabulate();
    Readinfo();
```

```
    Draw_Screen();
    Action();
    Setvideomode(TEXT_MODE);
}
```

程序流程为先进入 800×600 256 色绘图模式，然后使用 tabulate 函数计算并储存查表法绘点所需的数据，接着用 Readinfo 读取本模式的有关数据，包括屏幕宽高、字符宽高及行数等，在屏幕上以随机数画上 400 个任意圆(Draw_Screen)，并进入画面放大的功能(Action)，直到按 Esc 键则恢复文本模式，然后结束程序的执行。

```
void Draw_Screen()
{
    int cx, cy, r, color;
    int i;

    for(i=0, i<400;i++)  {
        cx  =random(600)+100;
        cy  =random(400)+100;
        color  =random(256);
        BCircle(cx, cy, r, color);
    }
}
void Readinfo()
{
    Screen_Width  =  peekb(0x0000, 0x044a);
    Screen_Rows   =  peekb(0x0000, 0x0484);
    Char_Height   =  peekb(0x0000, 0x0485);
    Screen_Height= Char_Height * Screen_Rows;
}
void Zoom(int xscale, int yscale, int sx, int sy)
{
    unsigned int code=0;
    int ex, ey;
    int maxx,maxy;

    xscale = (xscale<0) ? 0 : xscale; /* force xscale into range 0 to 7 */
    xscale = (xscale>7) ? 0 : xscale;
    yscale = (yscale<0) ? 0 : yscale; /* force yscale into range 0 to 7 */
    yscale = (yscale>7) ? 0 : yscalce;

    code |=xscale << 4; /* set x zoom factor */
    code |=yscale;       /* set y zoom factor */
    code |=0x80;         /* zoom  enable      */
    code = code << 8;
```

```
code |= 6;
outport(0x34,code); /* set zoom factor and ebable zoom */

maxx = Screen_Width - Window_Width * (xscale+1) -1;
maxy = Screen_Height- Window_Height * (yscale+1) -1;
sx = (sx<0)      ? 0      : sx;      /* force sx into 0 to maxx */
sx = (sx>maxx)   ? maxx   : sx;
sy = (sy<0)      ? 0      : sy;      /* force sy into 0 to maxy */
sy = (sy>maxy)   ? 0      : sy;

code = 0x1b;
code |=sx << 8;
outport(0x3d4, code);                /* set the x-start position */

ex = (xscale+1) * Window_Width+sx;
code = 0x1c;
code |= ex << 8;
outport(0x3d4, code);

code = 0x1d;
code = (sy & 0xff) <<8;
outport(0x3d4,code);

code = 0x1f;
code |= ( (sy &0xff00) <<3);
outport(0x3d4, code);

ey = (yscale+1) * Window_Height * Char_Height+sy;
code = 0x1e;
code |= (ey & 0xff) <<8;
outport(0x3d4, code);

outportb(0x3d4, 0x1f);
code = inportb(0x3d5);
code &= 0xf8;
code |= (ey & 0xff00) >>8;
outportb(0x3d4, 0x1f);
outportb(0x3d5, (char) (code & 0xff);

code = 0x20;
outportb(0x3d4, code);

code ++;
outportb(0x3d4, code);
```

```
    outportb(0x3d4, 0x23);
    code = inoortb(0x3d5);
    code &= 0xfb;
    outportb(0x3d4, 0x23);
    outportb(0x3d5, (char) (code & 0xff));
}

void BCircle(int, cx, int cy, int r, int color)
{
    void Put8cpixel(int, int, int, int)
    int x, y, d;

    x = 0;
    y = r;
    d = 3 - 2 * r;
    d0 {
        Put8cpixel(x, y, color, cx, cy);
        if( d < 0 )
                d = d+(x< <2)+6;
        else {
                d = d+(x-y)< <2)+10;
                y --;
        }
        x ++;
    } while(x<y);
    if(x= =y);
        Put8cpixel(x, y, color, cx, cy);
}
void Put8cpixel(int x, int y, int color, int cx, int cy)
{
    Writepixel256(cx+x, xy+y, color);
    Writepixel256(cx-x, xy+y, color);
    Writepixel256(cx+x, xy-y, color);
    Writepixel256(cx-x, xy-y, color);
    Writepixel256(cx-x, xy+y, color);
    Writepixel256(cx+x, xy-y, color);
    Writepixel256(cx-x, xy-v, color);
}
void Writepixel256(int x, int y, int color)
{
    unsigned offset;
    unsigned page;
    page = (table[y]+x) > > 16;
```

```
        offset = (table[y]+x) & 0xffff;

        Select__wpage(page);
        pokeb(0xa000, offset, color);
    }

    void tabulate()
    {
        int i;
        for(i=0; i<600; i++)
            table[i] = (long) 800 * i;
    }
```

上面这一段程序码是画点、画圆及建立表格的函数，前一节已经介绍过了，在此不再重述。

```
    void Readinfo()
    {
        Screen__Width  = peekb(0x0000, 0x44a);
        Screen__Rows   = peekb(0x0000, 0x0484) +1;
        Char__Height   = peekb(0x0000, 0x0485);
        Screen__Height= Char__Height * Screen__Rows;
    }
```

正如前面所说的：画面放大功能适合于各种绘图模式，因此在作放大之前必须先取得有关的参数才能正确地设定各寄存器的值，而这些参数就放在 BIOS 的数据区中。BIOS 数据区位于 0000:0400 开始的地方，其中和本节有关的部分列表如下：

表 5-6 与画面放大有关的 BIOS 数据区

位 址	长 度	说 明
0040:0049	1Byte	目前的图像模式
0040:004A	1Word	每行的字数
0040:004C	1Word	显示内存的大小
0040:004E	1Word	显示内存的起始偏移地址
0040:0062	1Byte	目前的显示页
0040:0084	1Byte	显示行数减 1
0040:0085	1Word	字符矩阵的高度

接着便是这个程序最重要的部分，即操作寄存器的实际情形：

```
    void Zoom(int xscale, int yscale, int sx, int sy)
    {
        unsigned int code=0;
        int ex, eyl;
```

```
int maxx, maxy;

xscale = (xscale<0) ? 0 : xscale; /* force xscale into range 0 to 7 */
xscale = (xscale>7) ? 7 : xscale;
yscale = (yscale<0) ? 0 : yscale; /* force yscale into range 0 to 7 */
yscale = (yscale>7) ? 7 : yscale;
```

我们将放大的倍数限制于 0–7 之间，以免超出指定的范围，产生不可预期的结果。

```
code |=xscale << 4;  /* set x zoom factor  */
code |=yscale;       /* set y zoom factor  */
code |=0x80;         /* zoom enable        */
code  =code << 8;
code |=6;
outport(0x3c4, code); /* set zoom factor and enable zoom */
```

这段程序码先将 X 轴放大倍数左移 4 个比特放入 code 变量，再将 Y 轴放大倍数一并存入，然后把第 7 个比特设为 1。接着，这些数据全部堆入高地址区(D8–D15)，再把寄存器的下标值 6 放入低地址区(D0–D7)，利用 outport 写入即可，整个过程如下图所示：（设 xscale = 5,yscale = 7）

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
-	-	-	-	-	-	-		-	1	0	1		-	-	-	code|=xscale<<4;
-	-	-	-	-	-	-	-	-	1	0	1	-	1	1	1	code|=yscale;
-	-	-	-	-	-	-	-	1	1	0	1	-	1	1	1	code|=0x80;
1	1	0	1	-	1	1	1	0	0	0	0	0	0	0	0	code|code<<8;
1	1	0	1	-	1	1	1	0	0	0	0	0	1	1	0	code|=6;

高地址区（D15–D8）　　低地址区（D7–D0）

```
maxx = Screen_Width - Window_Width  *(xscale+1) -1;
maxy = Screen_Height - Window_Height *(yscale+1) -1;

sx = (sx<0)      ? 0    : sx;   /* force sx into 0 to maxx */
sx = (sx>maxx)   ? maxx : sx;
sy = (sy>0)      ? 0    : sy;   /* force sy into 0 to maxy */
sy = (sy>maxy)   ? maxy : sy;
```

限制放大后区域的起始坐标值在 0–maxx 及 0–maxy 的范围内.

```
code=0x1b;
code|sx<<8;
outport(0x3d4,code)
```

设定起始 X 坐标值.

```
ex = (xscale+1) * Window__Width+sx;
code = 0x1c;
code |= ex << 8;
outport(0x3d4, code);
```

依照目前的屏幕宽度计算放大后区域的结束 X 坐标（右下角坐标）值，然后写入寄存器 3D4h Index 1ch.

```
code = 0x1d;
code |= (sy & 0xff) << 8;
outport(0x3d4, code);

code = 0x1f;
code |= ((sy & 0xff00) << 3);
outport(0x3d4, code);
```

将起始坐标（右上角坐标）的 sy 值写入寄存器. 由于 sy 值占两个寄存器，所以分成两个步骤，先写入 sy 值的低地址部分（sy & 0xff），将写入 sy 的高地址部分（sy & 0xff00）.

```
ey = (yscale+1) * Window__Height * Char__Height+sy;
code = 0x1e;
code |= (ey &0xff) << 8;
outport(ox3d4, code);

ouportb(0x3d4, 0x1f);
code = inportb(0x3d5);
code &= 0xf8;
code |= (ey & 0xff00) >> 8;
outportb(0x3d4, 0x1f);
outportb(0x3d5, (char) (code & 0xff);
```

这段的功能是将结束坐标（右下角）值 ey 写入寄存器，由于仍然是分两个部分（高地址及低地址）分别储存于两个寄存器，所以必须进行两次的写操作. 低地址部分由于寄存器 3D5，Index 1eh 为单独使用，可以直接写入，但是高地址部分则和 sy 的高地址部分共用，所以在写入之前须先用 inportb 函数读入暂存器 3D5，Index 1fh 的值，和新值组合后再行写入，以免覆盖旧值.

```
    .
    .
    .
code = 0x20;
outport(0x3d4, code);

code ++;
outport(0x3d4, code);

outportb(0x3d4, 0x23);
code = inportb(0x3d5);
code &= 0xfb;
outportb(0x3d4, 0x23);
outportb(0x3d5, (char) (code & 0xff));
```

最后将寄存器 3D5，Index 20h，Index 21h 分别存入 0，而 Index 23h 的第 3 个 bit(D2)也设为 0，则所有的设定操作即告完成。

```
void Action()
{
    int key;
    int quit=0;
    int xscale=2;
    int yscale=2;
    int x=0, y=0;

    while(!quit) {
        key = bioskey(0);
        switch(key) {
            cade UP :
                    y = (--y<0) ? 0 : y;
                    break;
            case DOWN :
                    y = (++y>Screen_Height) ? Screen_Height : y;
                    break;
            case LEFT :
                    x = (--x<0) ? 0 : x;
                    break;
            case RIGHT :
                    x = (++>Screen_Width : x;
                    break;
            case PLUS :
                    yscale = (++yscale>7) ? 7 :yscale;
                    break;
            case MINUS :
```

```
                yscale = (--yscale<0) ? 0 : yscale;
                break;
        case INS    :
                xscale = (++xscale>7) ? 7 : xscale;
                break;
        case DEL    :
                xscale = (--xscale<0) ? 0 : xscale;
                break;
        case ESC    :quit = 1;
                break;
        }
        Zoom(xscale, yscale, x, y);
    }
}
```

上面这个函数使我们可以利用键盘上的箭头键移动放大的区块、+、-键调整 Y 轴的放大倍数、Ins 及 Del 键调整、X 轴的放大倍数，按 Esc 键则可离开这个程序。执行程序就好像拿着一个可变倍的放大镜扫视屏幕一般地有趣。至于键盘常数(UP，DOWN…等)的定义则如 key.h 所示，利用这些常数的定义，可以使你的程序更容易地应用键盘，同时程序也更具可读性。

```
#define ESC        0x011b
#define LEFT       0x4b00
#define RIGHT      0x4d00
#define UP         0x4800
#define DOWN       0x5000
#define PLUS       0x4e2b
#define MINUS      0x4a2d
#define INS        0x5200
#define DEL        0x5300
```

以下便是这个放大程序 ZOOM.C 的完整列表:

```
/ * ZOOM.C * /

#include <dos.h>
#include <stdio.h>
#include <bios.h>
#include <time.h>
#include <stdlib.h>
#include "supervga.h"
#include "super.c"
#include "key.h"

#define ESC     0x011b
```

```
#define LEFT   0x4b00
#define RIGHT  0x4d00
#define UP     0x4800
#define DOWN   0x5000
#define PLUS   0x4e2d
#define MINUS  0x4a2d
#define INS    0x5200
#define DEL    0x5300

int Screen_Width;
int Screen_Height;
int Char_Height;
int Screen_Rows;
int Window_Width  = 2;
int Window_Height= 2;

void Writepixel256(int, int, int);
void BCircle(int, int, int, int);
void tabulate(void);
void Setvideomode(int);
void Zoom(int, int, int, int);
void Readinfo(void);
void Action(void);
void Draw_Screen(void);

long table[600];

main()
{
        Setvideomode(GC256_800x600);
        tabulate();
        Readinfo();
        Draw_Screen();
        Action();
        Setvideomode(TEXT_MODE);
}

void Action()
{
    int key;
    int quit=0;
    int xscale=2;
    int yscale=2;
    int x=0, y=0;
```

```
    while(!quit) {
        key = bioskey(0);
        switch(key) {
          case UP   :
                      y = (--y<0) ? 0 : y;
                      break;
          case DOWN :
                      y=(++y>Screen_Height) ? Screen_Height:y;
                      break;
          case LEFT :
                      x = (--x<0) ? 0 : x;
                      break;
          case RIGHT:
                      x=(++x>Screen_Width) ? Screen_Width:x;
                      break;
          case PLUS :
                      yscale = (++yscale>7) ? 7 : yscale;
                      break;
          case MINUS:
                      yscale = (--yscale<0) ? 0 : yscale;
                      break;
          case INS  :
                      xscale = (++xscale>7) ? 7 : xscale;
                      break;
          case DEL  :
                      xscale = (--xscale<0) ? 0 : xscale;
                      break;
          case ESC  : quit=1;
                      break;
    }
       Zoom(xscale, yscale, x, y);
    }
}

void Draw_Screen()
{
     int cx, cy, r, color;
     int i;

     for(i=0; i<400; i++) {
          cx = random(600) + 100;
          cy = random(400) + 100;
          color = random(256);
```

```
        r = random(100);
        BCircle(cx, cy, r, color);
    }
}

void Readinfo()
{
    Screen_Width  = peekb(0x0000, 0x044a);
    Screen_Rows   = peekb(0x0000, 0x0484) + 1;
    Char_Height   = peekb(0x0000, 0x0485);
    Screen_Height= Char_Height * Screen_Rows;
}

void Zoom(int xscale, int yscale, int sx, int sy)
{
    unsigned int code=0;
    int ex, ey;
    int maxx, maxy;

    xscale = (xscale<0) ? 0 : xscale;
    xscale = (xscale>7) ? 7 : xscale;
    yscale = (yscale<0) ? 0 : yscale;
    yscale = (yscale>7) ? 7 : yscale;

    code |= xscale << 4;  /* set x zoom factor */
    code |= yscale;       /* set y zoom factor */
    code |= 0x80;         /* zoom enable       */
    code = code << 8;
    code |= 6;
    outport(0x3c4,code); /* set zoom factor and enable zoom */

    maxx = Screen_Width - Window_Width * (xscale + 1) -1;
    maxy = Screen_Height - Window_Height * (yscale +1) -1;

    sx = (sx<0)    ? 0    : sx;  /* force sx into 0 to maxx */
    sx = (sx>maxx) ? maxx : sx;
    sy = (sy<0)    ? 0    : sy;  /* force sy into 0 to maxy */
    sy = (sy>maxy) ? maxy : sy;

    code = 0x1b;
    code |= sx << 8;
    outport(0x3d4, code);  /* set the x-start position */

    ex = (xscale + 1) * Window_Width + sx;
```

```
        code = 0x1c;
        code |= ex << 8;
        outport(0x3d4, code);

        code = 0x1d;
        code |= (sy & 0xff) << 8;
        outport(0x3d4, code);

        code = 0x1f;
        code |= ((sy & 0xff00) << 3);
        outport(0x3d4, code);

        ey = (yscale + 1) * Window_Height * Char_Height + sy;
        code = 0x1e;
        code |= (ey & 0xff) << 8;
        outport(0x3d4, code);

        outportb(0x3d4, 0x1f);
        code = inportb(0x3d5);
        code &= 0xf8;
        code |= (ey & 0xff00) >> 8;
        outportb(0x3d4, 0x1f);
        outportb(0x3d5, (char) (code & 0xff));

        code = 0x20;
        outport(0x3d4, code);

        code ++;
        outport(0x3d4, code);

        outportb(0x3d4, 0x23);
        code = inportb(0x3d5);
        code &= 0xfb;
        outportb(0x3d4, 0x23);
        outportb(0x3d5, (char) (code & 0xff));
}

void BCircle(int cx, int cy, int r, int color)
{
        void Put8cpixel(int, int, int, int, int);
        int x, y, d;

        x = 0;
        y = r;
```

```
        d = 3 - 2 * r;
        do {
            Put8cpixel(x, y, color, cx, cy);
            if( d<0 )
                d = d + (x<<2) + 6;
            else {
                d = d + ((x - y)<<2) + 10;
                y --;
            }
            x ++;
        } while(x<y);
        if( x==y )
            Put8cpixel(x, y, color, cx, cy);
}

void Put8cpixel(int x, int y, int color, int cx, int cy)
{
        Writepixel256(cx+x, cy+y, color);
        Writepixel256(cx-x, cy+y, color);
        Writepixel256(cx+x, cy-y, color);
        Writepixel256(cx-x, cy-y, color);
        Writepixel256(cx+y, cy+x, color);
        Writepixel256(cx-y, cy+x, color);
        Writepixel256(cx+y, cy-x, color);
        Writepixel256(cx-y, cy-x, color);
}

void Writepixel256(int x, int y, int color)
{
        unsigned offset;
        unsigned page;

        page = (table[y] + x) >> 16;
        offset = (table[y] + x) & 0xffff;

        Select_wpage(page);
        pokeb(0xa000, offset, color);
}

void tabulate()
{
      int i;

      for(i=0; i<600; i++)
```

```
        table[i] = (long) 800 * i;
}
```

一般而言，用户在购买 Tseng Super VGA 卡时，通常都可以在厂商处获得一张显示及工具磁盘，其中就有一些放大画面的驻留工具程序，利用这些驻留程序，就可以在大部分绘图程序上做局部放大的效果．然而既然我们已经知道如何自行做画面放大的功能，我们就可以自行编制这些程序了．第七章我们将会把这个程序变为驻留程序，使得这个程序的应用范围更广、更好用．

5.6 结束语

在应用 Super VGA 绘图时，由于图形数据量很大，所以非常重视各种方法的效率问题．利用 BIOS 的功能调用虽然方便，但是效率及速度却往往不合人意．因此，凡是执行次数较多的函数必须自行设计，直接存取内存及寄存器来发挥硬件的最大功能，因此多了解各寄存器的功能便成为更进一步研究 Super VGA 的重要课题．

另外，由于 Super VGA 适配器中有许多寄存器负责扫描屏幕的宽高及时序控制，如果不小心写入错误的值，可能会造成屏幕"哔"地一声全然变黑，看不到任何讯息．此时只需将屏幕的开关切掉再重开即可；当然这种情形不要经常发生，不然可能有损屏幕的寿命，所以在写入寄存器时要特别小心检查．

本章只是提供读者对 Super VGA 程序一个初步的认识，读者如果有兴趣作更进一步的研究，则请自行参考 Super VGA 的相关书籍．

第六章　绘图与打印图形程序设计

6.1　图形打印的基本原理

6.1.1　Turbo C 的打印函数

通常我们在设计一个绘图程序之后，除了将自己精心编写的图形显示在屏幕上之外，当然也会想到利用打印机将图形打印在打印纸上以供应用。本节将针对与 EPSON 兼容的单色点阵式打印机详细说明如何使用 Turbo C 编写各种绘图模式下的画面打印常用程序。首先，让我们先来看看 Turbo C 有哪些支持打印机的函数：

定义：#include<stdio.h>

　　　int fprintf(FILE *stream,const char *format_string…);

调用：fprintf(stdprn,"%s\n\r",string);

　　　fprintf(stdprn,"%c",ch);

fprintf()函数实际上并不是支持打印机的函数，然而由于本函数可以指定数据输出的文件指针，而 DOS 又将标准的输出装置（如屏幕、键盘、打印机…等）视为一般的文件，所以我们可以利用这个函数将格式化的数据输出至打印机。所需要做的只是将该函数的第一个参数指定为标准打印机输出 Stdprn 即可；使用之前并不需利用 fopen()等函数打开文件，因为一载入 DOS 就已经帮你启动好了。

fprintf()是属于高级的函数，用起来虽然方便，却不符合程序的需求。例如，我们无法利用本函数来得知目前打印机所处的状态。所以 Turbo C 又提供了一个较为低级的打印函数 biosprint()，现在将这个函数说明如下：

定义：#include<bios.h>

　　　int biosprint(int mode,int data,int printer);

调用：biosprint(0,'C',0);

　　　biosprint(1,'C',LPT1);

说明：mode=0：将data的低字节送至打印机

　　　mode=1：初始化打印机

　　　mode=2：返回目前打印机的状态

　　　printer=0表示LTP1；1表示LPT2，您也可以直接使用常数LPT1，LPT2。

biosprint()顾名思义，是直接对应到 BIOS 的中断 17h 的一个函数，所以在程序中使用下列这两段程序码，所达到的功能都是相同的。

```
                              union REG r;
biosprint(0,'C',0);           r.h.ah = 0;
                              r.h.al = 'C';
                              r.x.dx = 0;
                              int86(0x17, &r, &r);
```

因为 INT 17h 的功能和 biosprint()的功能相同，所以接着顺便将 INT17h 的功能说明罗列如表 6–1 以供读者参考。

表 6–1　INT 17h 打印机功能调用表

FUNCTION 0:	打印一个字符
输入值:	AH＝0
	AL＝要打印字符的ASCⅡ码
	DX＝打印机编号（0＝LPT1，1＝LPT2···）
返回值:	AH＝打印机目前状态
FUNCTION 1:	初始化打印机
输入值:	AH＝1
	DX＝打印机编号
返回值:	AH＝打印机目前状态
FUNCTION 2:	读取打印机目前状态
输入值:	AH＝2
	DX＝打印机编号
返回值:	AH＝打印机目前状态

此外，不论 INT 17h 功能调用或是 biosprint()函数，在调用完毕后，都会返回一个打印机目前的状态字节，其中 INT 17h 是放在 AH 中，而 biosprint 则是放在返回整数的低字节中，其说明如表 6–2:

表 6–2　INT 17h 打印机状态字节说明

bit 7＝1:	打印机不忙
bit 6＝1:	选择接收模式
bit 5＝1:	缺纸
bit 4＝0:	Off–Line
bit 3＝1:	I/O错误
bit 2＝?:	保留
bit 1＝?:	保留
bit 0＝1:	超时错误

6.1.2　文字打印程序

了解这个状态字节所代表的意义之后，程序中便可以利用功能 2 来取得目前打印机的状态，而不致于盲目地操作打印机，造成数据丢失或程序执行过去了。

值得一提的是：bit 0 的超时错误是表示 BIOS 经过多次的连续尝试对打印机写入数据之后，而打印机却一直处于忙碌状态才会发生的错误。如果你的打印机在正常的使用下常常发生逾时的错误，表示 BIOS 对打印机的检查循环次数太少了，你可以增加内存地址 0040:0078 中的值，让 BIOS 做更多次的检查及尝试。这个地址的预设值是 14h(20)，并不表示 BIOS 只做 20 次的检查，而是 $20 \times 4 \times 65536 = 5242880$ 次的检查。

现在让我们写一个程序 P_TEST.C，输出一段文字到打印机上：

```
#include <stdio.h>
#include <dos.h>

int  p_status(void);
void p_putc(char);
void p_puts(char *);
int  p_notbusy(void);

main()
{
    while(!p_notbusy());
    p_puts("*  This is a test program for printer  *\n\r");
    p_puts("Put a char : ");
    p_putc('C');
    p_puts("\n\r");
}

int p_status()
{
    union REGS r;

    r.h.ah = 2;
    r.x.dx = 0;
    int86(0x17, &r, &r);
    return(r.h.ah);
}

void p_putc(char ch)
{
    fprintf(stdprn, "%c", ch);
}

void p_puts(char *str)
{
    fprintf(stdprn, "%s", str);
}
```

```
int p__notbusy()
{
    union REGS r;

    r.h.ah = 2;
    r.x.dx = 0;
    int86(0x17, &r, &r);
    return((r.h.ah & 0x80) >> 7);
}
```

P__TEST.C 执行后会在打印机上打出如下字样:

```
* This is a test program for printer *
put a  char : C
```

6.1.3 图形打印原理

由于以上打印程序所打印的数据是属于文字数据，所以我们所做的只是单纯地利用 fprintf()函数，将文字数据送至打印机打出即可。然而如果要对打印出的字形加一点变化(如斜体、粗体等)，或是要打印图形，那么必须在要打印的数据送出之前，先告诉打印机要怎样打印稍后会送来的数据．一般计算机都是以控制命令来与打印机沟通的．以 EPSON 系列或兼容的打印机为例，因为它的命令完全是以 ESC 开头，所以称为 Escape Sequence Code。Escape Sequence Code 的数目多达数十个，在此无法一一介绍，仅就与图形输出有关的部分加以说明，如果读者有兴趣的话，可参考打印机手册上更进一步的说明．下面就是几个和图形输出有关的打印机命令:

表 6-3 与图形输出有关的打印机控制命令

<ESC> 'K' n1 n2 m1 m2…	打印单密度为 8 位的图形
<ESC> 'L' n1 n2 m1 m2…	打印单密度为 8 位的图形
<ESC> 'Y' n1 n2 m1 m2…	双倍速度打印双密度为 8 位的图形
<ESC> 'Z' n1 n2 m1 m2…	打印四倍密度的 8 位图形，设定 1／8 英寸的行距
<ESC> '0'	设定1／8英寸的行距
<ESC> '1'	设定7／72英寸的行距
<ESC> '2'	设定1／6英寸的行距
<ESC> '2'n	设定n／216英寸的列距
<ESC> 'A'n	设定n／72英寸的行距
n1=WIDTH%256	(打印数据宽度取余数)
n2=WIDTH／256	(打印数据宽度取商)
m1, m2…	(要打印的数据)
n	(整数)

程序中要做图形输出时，必须从<ESC>'K' 'L' 'Y' 'Z' 4个命令中挑选出一个你要使用的打印密度，然后通过n1及n2两个参数告诉打印机要传送多少位的数据。n2为所要传送字节数目（在此即为图形的宽度）除以256所得到的商，而n1则为其余数。假设我们想打印720个字节的数据，则应以如下的顺序送至打印机：

<ESC>'L'208 2 m1 m2 . . .

图形数据

上列数据如果写成C语言实际的程序码，则应该是：

```
#define ESC 27
p_putc(ESC);
p_putc('L');
p_putc(208);
p_putc(2);
p_putc(m1);
p_putc(m2);
.
.
.
```

至于如何将图形整理成打印机所能接受的打印方式呢？首先我们必须了解：打印机所组织的图形和真正屏幕上视频内存所储存的图形数据组织的方式并不相同，请参考图6-1：

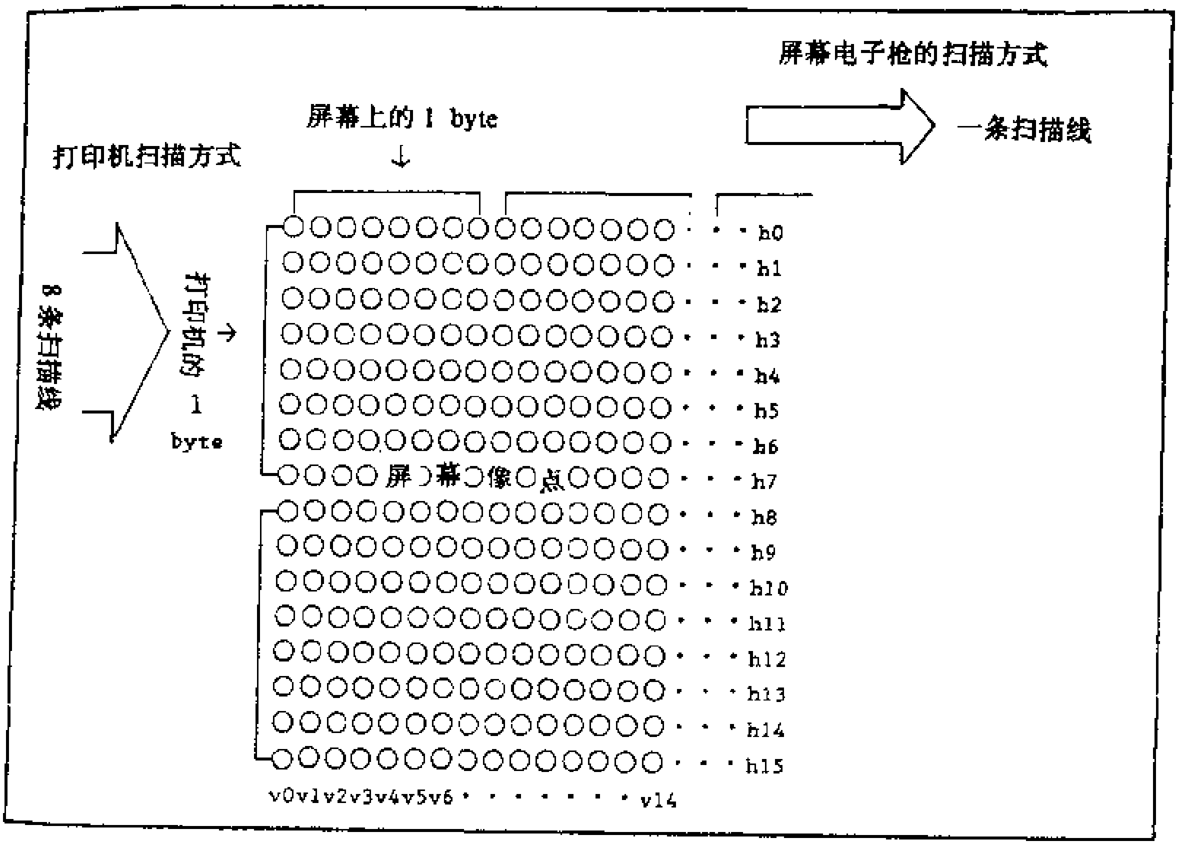


图6-1 打印机的图形打印方式

如图 6–1 所示，在屏幕水平方向每 8 点组成 1 个字节。而对打印机而言，由于打印机的撞针是采用垂直排列的，所以传送给打印机的数据必须先经过转换，将 V0 行的 h0–h7 包装成 1 个字节（原本是 8 个独立的字节的最左字节）成为送到打印机的第一个打印数据字节，而 V1 行的 h0–n7 包装成第二个字节，依此类推。转换的程序码如下所列：

```
for(i=sy; i<ey; i+=8)
{
    p_putc(ESC),
    p_putc('L');
    p_putc(n1);
    P_putc(n2);
    for (j=sx; j<ex; j++)  {
       data  =  0;
       for (k=0; k<8; k++);
           if ( getpixel (j, i+k) );
               data |= ( 1<< (7-k) );
           p_putc(data);
    }
```

这个程序假定所要打印的区块为(sx,sy)–(ex,ey)，其中 i 变量记录目前扫描线（即 y 坐标），由于打印机必须同时将 8 条扫描线的数据收集成 1 字节，所以 i 变量每次加 8，而 j 变量则是记载目前的 x 坐标，K 循环即是执行收集的功能。上述这三个循环及各变量的关系，可以用图 6–2 来解释：

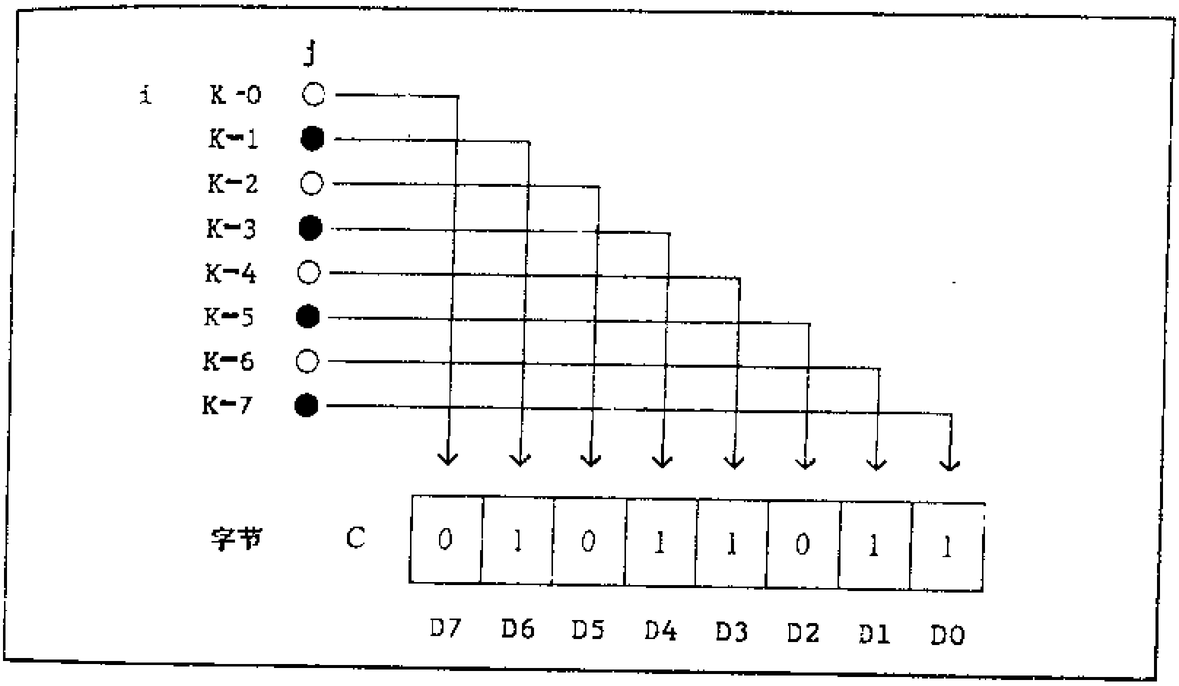


图 6–2 屏幕与打印机的图点排列关系

6.2 单色打印图形程序设计

根据上述的转换方法，我们写了一个单色的屏幕硬拷贝(Hard Copy)程序MPTRSCR.C，以下就是程序的列表:

```
/ * MPTRSCR.C *

#include <stdio.h>
#include <dos.h>
#include <graphics.h>
#include <stdlib.h>

#define ESC     27
void p_putc(char);
void p_puts(char *);
void Hercules_printscr(int, int, int, int);

main()
{
     int graph_driver=HERCMONO, graph_mode=HERCMONOHI;
     register int i:

     detectgraph(&graph_driver, &graph_mode);
     if(graph_driver!=HERCMONO) {
          puts("This program requires a Mono-Monitor with Hercules card");
          exit(0):
      }

     initgraph(&graph_driver, &graph_mode, "");
     for(i=40; i<=160; i+=10)
            circle(360+i,150, i);
     for(i=160; i>=40; i-=10)
            circle(360-i,150, i);
     settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);
     outtextxy(150,300,"Turbo C Screen Dump Demo.");
     Hercules_printscr(0,0,719,347);
     getch();
     closegraph();
}

void p_putc(char ch)
{
      fprintf(stdprn, "%c", ch);
```

```
}

void p__puts(char * str)
{
        fprintf(stdprn, "%s", str);
}

void Hercules__printscr(int sx, int sy, int ex, int ey)
{
        int deltax;
        int n1, n2;
        int i, j, k;
        char data;

        deltax = ex - sx + 1;
        p__putc(ESC);
        p__putc('3');
        p__putc(20);
        n1 = deltax % 256;
        n2 = deltax / 256;
        for(i=sy; i<=ey; i+=8) {
                p__putc(ESC);
                p__putc('L');
                p__putc(n1);
                p__putc(n2);
                for(j=sx; j<=ex; j++)     {
                        data = 0;
                        for(k=0; k<8; k++)
                        if(getpixel(j, i+k))
                            data |= (1<<(7-k));
                        p__putc(data);
                }
                p__putc('\n');
                p__putc('\r');
        }
        p__putc(ESC);
        p__putc('2');
}
```

这个程序首先执行硬件检查的功能，看看计算机上配备的是不是单色的绘图适配器，如果不是的话则拒绝继续执行，并输出有关信息，以免打错数据。如果电脑上确实使用的是单色绘图适配器，则程序进入单色绘图模式，接着在屏幕上画一张简易的图形，并用 Hercules__printscr 函数将图形打印出，如图 6-3 所示。

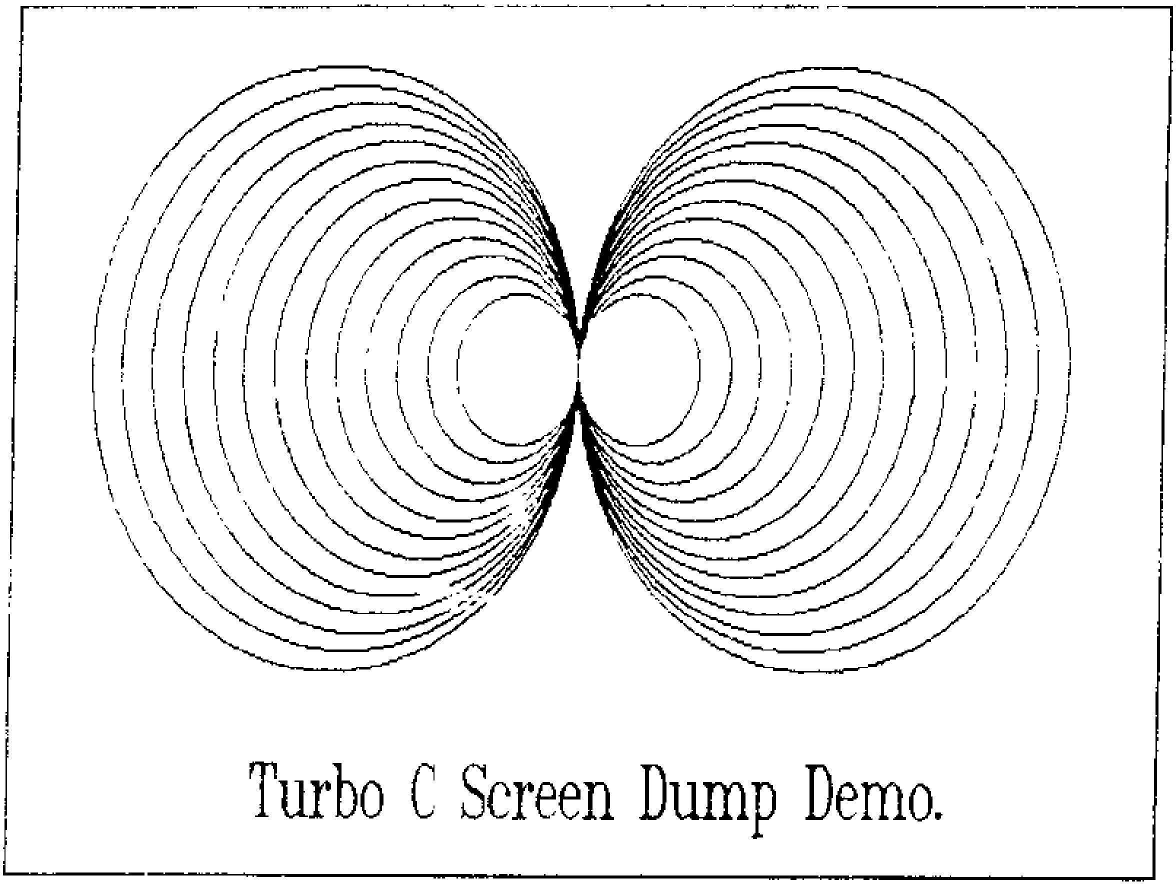


图 6-3　程序 MPTRSCR.C 的执行结果

Hercules_printscr 函数可以由 sx，sy，ex，ey 等 4 个参数来决定打印的范围。在打印之前，必须先使用：

<ESC>　<'3'>　<n>

命令调整打印机的行距，以免打印出来的图形太长或太扁。n 值的大小视所使用的打印机及打印密度而定（当然必须是能接受 Esc-sequence 命令的），笔者所使用的 Canon BJ-10c 喷墨打印机以 n＝20 最为适当，若读者使用别种形式的打印机，这个值可能就要自己调整了。在调整行距命令后，紧接着的就是按位图形打印命令：

<ESC>　<'L'>　n1 n2

以每 8 条扫描线为一组，输出一个<ESC>　<'L'>　n1 +n2 命令，后面则是跟着转换后要打印的数据，应该有 ex-sx+1 个字节，亦即 n2×256+n1 个字节。打印完毕后请记得使用<ESC><'2'>命令恢复原来的列距，以免造成麻烦。

现在，我们再以另一个 biosprint()函数编写简单的打印程序 PRINT2.C。这个程序的使用格式如下：

C>print2　<filename>　<mode>

其中：filename 是我们所要打印的图形文件名，只要图形文件的格式与本书第三章中的 TPC，PUT，MAP，BPC 等格式相同，均可使用 PRINT2.C 程序打印出。

mode 是打印的模式，如果输入 0，则图形以反白打出。如果输入其它参数，则图形

正常印出，为了简化程序，我们在程序中将图形打印的范围予以固定，这样程序比较容易了解，但也降低了执行的效率，读者如果有兴趣，可以增强它的功能。以下便是 PRINT2.C 程序的列表：

```
/ * PRINT2.C * /

#include "graphics.h"
#include "stdio.h"
#define ESC 27
unsigned x,y,file_length;
char buff[32000];
main(int argc,char * argv[] )
{
    FILE * fp;
    char * buff;
    int driver=DETECT,mode=0,t;

    if( argc< 2 ) {
        printf("\nUsage : print1 <Graph_file> < Mode >");
        printf("\nMode     = 1 : Normal           ");
        printf("\n         = 0 : Reverse          ");
        exit(0);
    }
    initgraph(&driver,&mode,"");
    fp=fopen( argv[1],"rb");

    x = getw( fp);
    y = getw (fp);
    t = x % 8;
    if( t>0 )
        x= x - t + 8;
    file_length = ( x / 8) * y + 4;
    buff=(char * )malloc( file_length );
    fseek(fp,0,SEEK_SET);
    fread( buff ,1,file_length, fp);
    fclose(fp);

    if( atoi(argv[2]) == 0 )
        putimage( 0,0, buff, 0 );
    else  putimage( 0,0, buff, 4 );

    print_screen( * (buff), * (buff+1) ,y );
    free(buff);
    closegraph();
```

```
41:}
42:
43:print__screen()
44:{
45:unsigned   row,col,p,h;
46:
47:biosprint(0,ESC,0);
48:biosprint(0,51,0);
49:biosprint(0,24,0);
50:
51:for(row=0;row<43;row++)        {
52:        biosprint(0,ESC,0);
53:        biosprint(0,'Y',0);
54:        biosprint(0, 208,    0);
55:        biosprint(0, 2, 0):
56:
57:        p=0;
58:        for( col=0;col<=719;col++ )        {
59:              for( y= row*8; y<=row*8+7; y++ ) {
60:                 p=p<<1;
61:                 if( getpixel(col,y) )
62:                        p++;
63:              }
64:              biosprint(0,p,0);
65:        }
66:        biosprint(0,13,0);    /* carriage return */
67:        biosprint(0,10,0);    /* line feed */
68:}
69:  biosprint(0,ESC,0);
70:  biosprint(0,'A',0);
71:  biosprint(0,12,0);
72:  biosprint(0,ESC,0);
73:  biosprint(0,2,0);
74:}
```

程序说明:

14–19　　如果输入参数少于2，则输出使用说明.

20　　进入绘图模式.

23–32　　根据图形文件最前面4 byte决定图形的大小.

47–49　　设定打印行距，此处以Canon BJ–10e喷墨打印机为例，选择24/216 inch.

51–68　　打印一水平线（8点一起打印).

52–55　　设定水平720点，双倍密度打印.

66–67　打印机换行及置于最左.

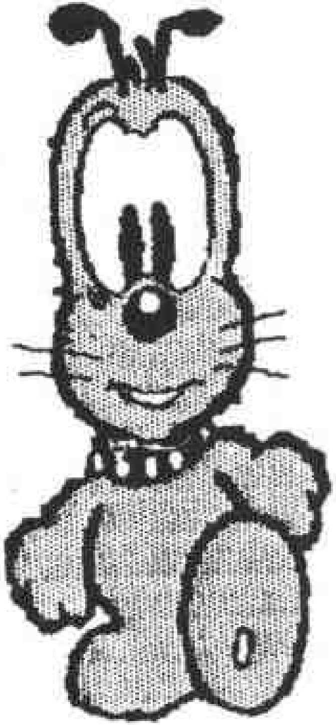
69–73　恢复打印机原来的设定状态.

图 6–4　PRINT2.C 程序的执行实例

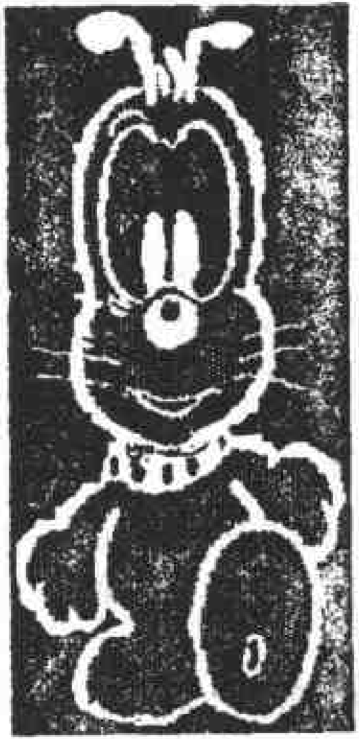
6.3　彩色图形打印

和前一个程序同样的原理，我们也可以写一个简易的彩色 EGA／VGA 图像硬拷贝程序 CPTRSCR.C，其列表如下：

```
/ *  CPTRSCR.C  * /

#include  <stdio.h>
#include  <dos.h>
#include  <graphics.h>
#include  <stdlib.h>

#define ESC      27

void p_putc(char);
void p_puts(char  * );
void EGAVGA_printscr(int, int, int, int, int);

main()
{
     int graph_driver=DETECT, graph_mode;
```



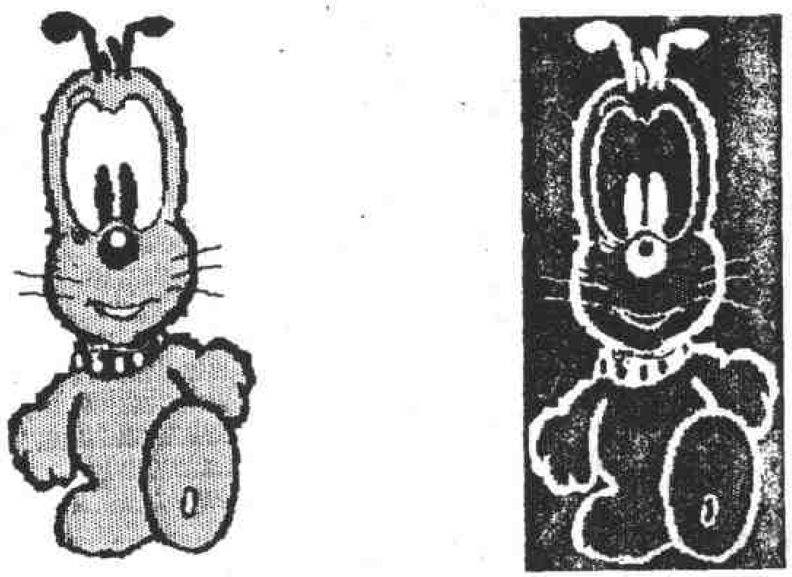
(a) 正常打印



(b) 反白打印

66—67　　打印机换行及置于最左.

69—73　　恢复打印机原来的设定状态.



(a) 正常打印　　(b) 反白打印

图 6—4　PRINT2.C 程序的执行实例

6.3　彩色图形打印

和前一个程序同样的原理，我们也可以写一个简易的彩色 EGA／VGA 图像硬拷贝程序 CPTRSCR.C，其列表如下：

```
/ *  CPTRSCR.C  * /

#include <stdio.h>
#include <dos.h>
#include <graphics.h>
#include <stdlib.h>

#define ESC      27

void p_putc(char);
void p_puts(char *);
void EGAVGA_printscr(int, int, int, int, int);

main()
{
    int graph_driver=DETECT, graph_mode;
```

```
    register int i;

    detectgraph(&graph_driver, &graph_mode);
    switch(graph_driver) {
        case VGA : graph_mode = VGAMED;
                   break;
        case EGA : graph_mode = EGAHI;
                   break;
        default  :
         puts("This program requires a Color-Monitor with EGA/VGA card");
               closegraph();
               exit(1);
    }

    initgraph(&graph_driver, &graph_mode, "");
    setcolor(1);            /* Blue   */
    for(i=40; i<=150; i+=10)
      circle(300+i,150, i);
    setcolor(14);           /* Yellow */
    for(i=150; i>=40; i-=10)
      circle(300-i,150, i);
    setcolor(15);           /* White  */
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);
    outtextxy(85,280,"Turbo C Screen Dump Demo.");
    setcolor(13);
    outtextxy(270,320, "Color");
    EGAVGA_printscr(0,0,639,349,0);
    getch();
    closegraph();
}

void p_putc(char ch)
{
      fprintf(stdprn, "%c", ch);
}

void p_puts(char *str)
{
      fprintf(stdprn, "%s", str);
}

void EGAVGA_printscr(int sx, int sy, int ex, int ey, int bkcolor)
{
   int deltax;
```

```
    int n1, n2;
    int i, j, k;
    char data;

    deltax = ex - sx + 1;
    p_putc(ESC);
    p_putc('3');
    p_putc(20);
    n1 = deltax % 256;
    n2 = deltax / 256;
    for( i=sy; i<=ey; i+=8 ) {
      p_putc(ESC);
      p_putc('L');
      p_putc(n1);
      p_putc(n2);
      for(j=sx; j<=ex; j++)     {
            data = 0;
            for(k=0; k<8; k++)
               if(getpixel(j, i+k)!=bkcolor)
                 data |= (1<<(7-k));
            p_putc(data);
      }
      p_putc('\n');
      p_putc('\r');
  }
  p_putc(ESC);
  p_putc('2');
}
```

EGAVGA_printscrc 和 Hercules_printscr 函数不同的是：在彩色 16 色的环境下背景色不再固定为黑色，所以 EGAVGA_printescr 函数必须多加上一个参数，指定背景色为多少；这样，打印机便将背景色视为 0，而其它颜色全部视为 1 打印．CPTRSCR.C 的执行结果如图 6-5 所示：

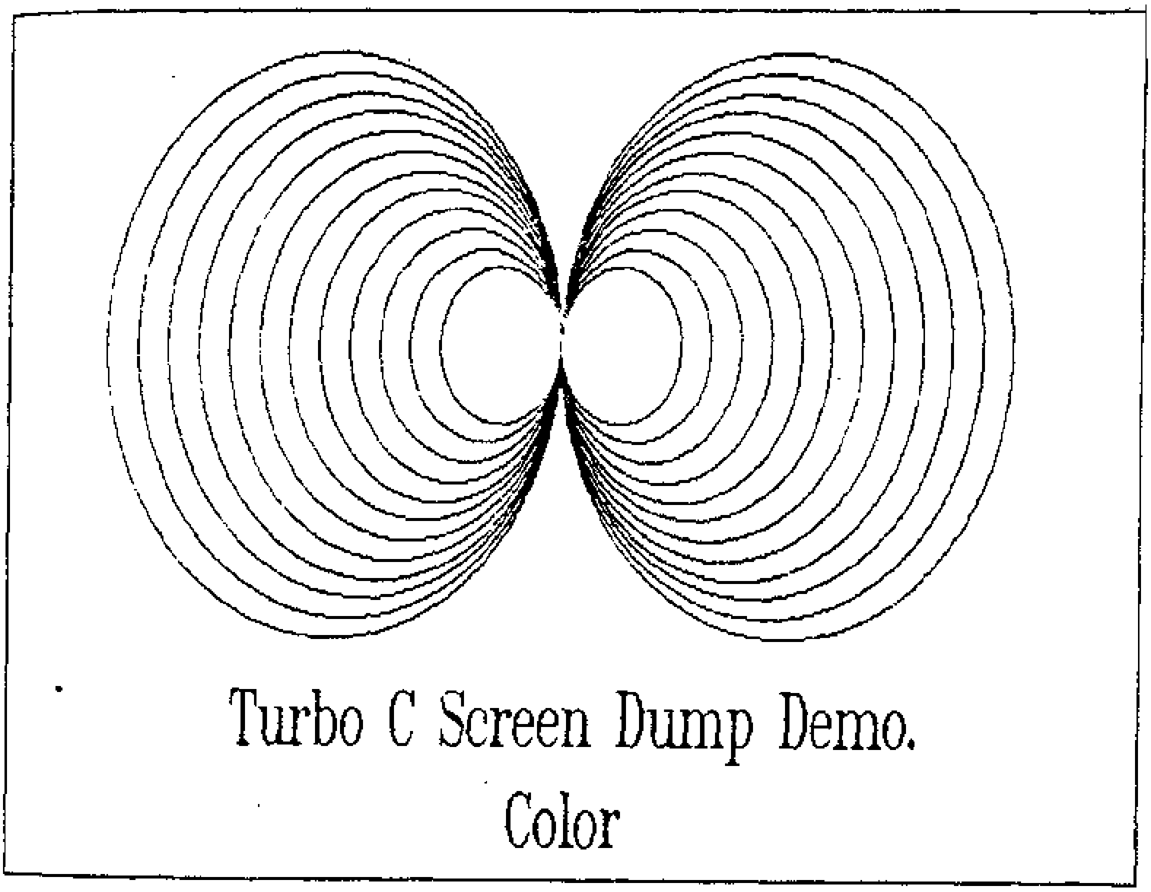


图 6-5 CPTRSCR.C 程序的执行结果

6.4 绘图程序设计

不管基于什么理由，你一定曾经想过要将画面上的全部或部分图形储存成一个图形文件以供使用，本节就是要探讨这些技术。我们希望在本节所设计的程序具备基本的交互式设定区块的功能外，还要有显示目前区块相关信息，并可对区块做反相、变亮、变暗、镜射等效果，至于驻留部分，则留待下一节再行介绍。

一般而言，所谓的绘图程序皆是指一驻留程序，可在任何绘图模式下利用某些特殊键（热键）将程序的驻留部分调出，它首先暂停前台程序的执行，然后截取原来的程序。除了存图外 某些绘图程序也提供区块打印及画面变化的功能。本节所编写的绘图程序提供了一个函数 mgetpic，调用本函数即进入绘图的模态，用户可利用箭头键及一些特殊键达到区块存图、打印及画面变化等功能。由于它还不是一个驻留程序，所以仅能以 C 语言来调用使用，运用范围可能不太普遍；然而你不妨将这个程序视为驻留绘图程序的前身，先熟悉绘图程序的一般设计技巧，在第 7 章中，我们将会把它修改成为实用的驻留绘图程序。

设计绘图程序之前，必须先确定这个程序要达到哪些功能，并确定所对应的键。我们对绘图程序编程如下：

表 6-3　绘图程序规定

	程序功能	按键
1	可自由决定区块大小	箭头键及空格键
2	可随时离开	ESC 键
3	可将区块储存成 *.TPC 文件	Alt-S
4	可将图形载入缓冲区	Alt-G
5	可将缓冲区内图形秀出	Alt-D
6	可显示目前区块的相关讯息	Alt-I
7	可将区块内图形反白	Alt-R
8	可将 *.TPC 的图形文件载入	Alt-L
9	可将区块变亮	Alt-T
10	可将区块变暗	Alt-K
11	可将区块图形镜射	Alt-M
12	可将区块图形上下颠倒	Alt-F
13	可打印区块	Alt-P

确定了上述各项功能后，接着就让我们来详细地介绍各部分的程序设计技巧，首先是键定义部分：(KEY.H)

```
#define ESC      0x011b
#define LEFT     0x4b00
#define RIGHT    0x4d00
#define UP       0x4800
#define DOWN     0x5000
#define PLUS     0x4e2b
#define MINUS    0x4a2d
#define INS      0x5200
#define DEL      0x5300
#define SPACE    0x3920
#define BACK     0x0e08
#define ENTER    0x1c0d
#define Alt-S    0x1f00
#define Alt-D    0x2000
#define Alt-P    0x1900
#define Alt-I    0x1700
#define Alt-R    0x1300
#define Alt-L    0x2600
#define Alt-K    0x2500
#define Alt-T    0x1400
#define Alt-M    0x3200
#define Alt-F    0x2100
#define Alt-G    0x2200
```

这个包含文件将所有本程序要用到的按键、常数全部收集在一起，这样可增加主程序的可读性，使用起来也较为方便。

程序一开始，是将所有需要的包含文件包括进来，同时并对程序中所需要使用的常数(包括屏幕最大宽度及高度、逻辑真值、假值）加以定义：

```
#include <graphics.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include "key.h"

#define  TRUE         1
#define  FALSE        0
#define  ON           1
#define  OFF          0
#define  POST         0
#define  DELTA        1
#define  MaxX         720
#define  MaxY         348
#define  NORMAL       0
#define  REVERSE      1
```

就如前文所说的，本节所介绍的绘图程序只是一个名为 mgetpic 的主要函数，这个函数和市面上绘图软件的部分功能相似，其写法如下：

```
void mgetpic()
  {
  int x=0, y=0, ox=0, oy=0;
  int deltax=10, deltay=10, odeltax=10, odeltay=10;
  int d=1;
  int key;
  int quit=FALSE;
  int IncDecFlag=POST;
  int DispInfo=OFF;
  int i;
  char *buf, *grpdata=NULL;
  void RectXor(int, int, int, int);
  void Light(int, int, int, int);
  void Dark(int, int, int, int);
  void Reverse(int, int, int, int);
  void Getfilename(char *);
  void Saveimage(char *, int, int, int, int);
  void Loadimage(char *, int, int);
```

```
void Flipflop(int, int, int, int);
void Mirror(int, int, int, int);
void Hercules__printscr(int, int, int, int);
```

mgetpic 函数首先定义函数中所需使用到的变量及函数变量，x，y 及 deltax，deltay 分别记录按箭头键更动区块后，最新的区块左上角坐标及 x，y 轴的长度，而 ox，ox 和 odeltax，odeltay 则保留着更动前的坐标及长度，d 是目前按箭头键区块移动或改变大小的增量，其值愈大，区块移动也愈快. d 的值为 1，亦即按一次箭头键，区块移动一个像素点. 这些变量的关系，可以由图 6-6 来表示：

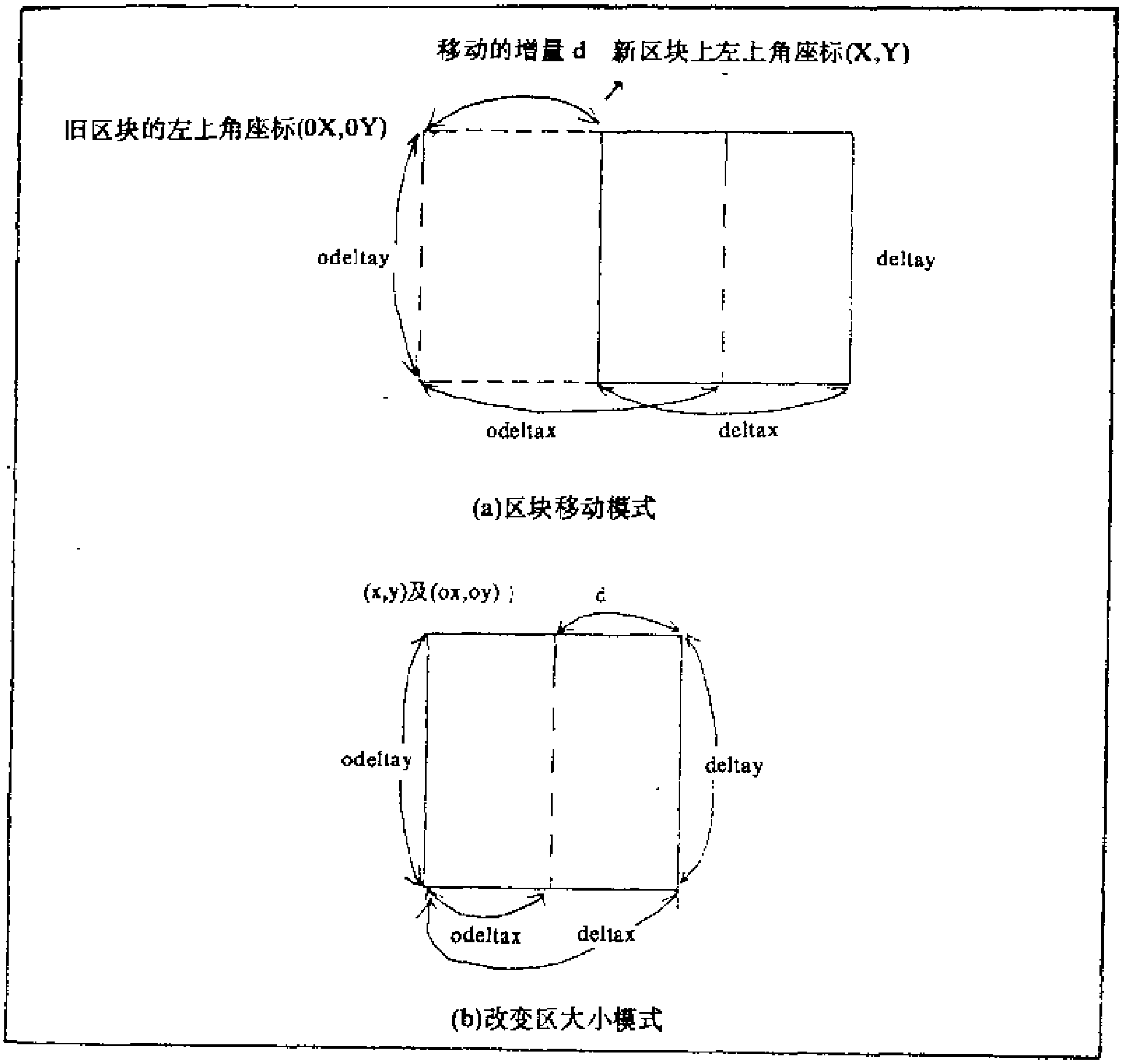


图 6-6　区块的改变与坐标变量的关系

在图 6-6 中，虚线的框为原来的方框，而实线的框则为用户按下右键后所可能产生的两种情形.

mgetpic 函数中定义了 quit 变量以判断是否要脱离循环的标志. IncDecFlag 变量则说明目前按箭头键所产生的影响应该是改变区块的大小（改变 deltax，deltay 之值），还

是移动区块的位置（改变 x，y 的值）；如果 IncDecFlag 等于 DELTA 则为前者，若是 POST，则为后者。DispInfo 变量指明在按箭头键移动区块或改变区块大小的过程中是否要显示区块目前的位置、宽高及信息于屏幕左上角位置。字符指针变量 buf 及 grapdata 则分别指向储存屏幕左上角背景数据、暂存区块图形数据的缓冲区开头。其它定义的函数功能则如下表所列：

表 6-5 MGETPIC.C 程序中的函数

函　　数	功　　能
RectXor	用 XOR 的方式画一矩形
Light	将区块图形变亮
Dark	将区块图形变暗
Reverse	将区块图形反白
Getfilename	取得文件名称
Saveimage	以 *.TPC 格式储存图形
Loadimage	读取图形文件
Hipflop	将区块图形上下颠倒
Mirror	对区块图形做镜射效果
Hercules-printscr	区块图形打印

定义部分结束，紧接着便是程序部分：

```
buf=(char *) malloc(imagesize(0,0,216,9));
getimage(0,0,216,9,buf);
RectXor(x,y,deltax,deltay);
```

这段程序先保存屏幕左上角部分(0,0)-(216,9)的背景，以防止原画面遭到破坏（因为后面的程序会利用此区域做输出及交谈对话），然后在左上角画上第一个区块指示方框。

```
while(!quit){
key=bioskey(0);
switch(key){
```

程序进入读取按键的循环，利用 bioskey(0)来等待并读取键盘的按键，将读到的键值交由变量 key 储存，再以 key 之值来判断用户按了什么键：

```
case ESC:quit=TRUE;
        break;
```

如果使用者按了 ESC，则令 quit 变量等于 TRUE，便可离开 while 循环。

```
case UP :if(IncDecFlag==POST)
         {
         y-=d;
         y=(y<0)?0 :y;
         }
```

```
else
{
deltay-=d;
deltay=(deltay<1) ? 1 : deltay;
}
break;
```

如果按的是向上键，那么得看看 IncDecFlag 变量目前所指示的是移动位置(IncDecFlag == POST)，还是改变区块大小(IncDecFlag == DELTA)；如果 IncDecFlag==POST，则改变 y 的值，否则改变 deltay 的大小，其余如按下左右键也是同样的方法：

```
case DOWN :if(IncDecFlag=POST)
        {
          y+=d;
          y=(y+deltay)>MaxY) ? y-d :y;
        }
        else
        {
          deltay +=d;
          deltay =((deltay+y)> =MaxY) ? deltay-d : deltay;
        }
    case RIGHT: if(IncDecFlag= =POST)
        {
        x += d;
        x = ((x+deltax)> =MaxX) ? x-d : x;
        }
        else
        {
        deltax += d;
        deltax = ((deltax+x)> =MaxX) ? deltax-d : deltax;
        }
        break;
  case LEFT : if(IncDecFlag= =POST)
        {
        x -= d;
        x = (x<0) ? 0 : x;
        }
        else
        {
        deltax -= d;
        deltax = (deltax<1) ? 1 : deltax;
        }
        break;
```

```
case SPACE: IncDecFlag = (IncDecFlag= =POST) ? DELTA : POST;
            break;
```

按空格键可以改变按箭头键的模式，其原理只不过是对 IncDecFlag 这个变量做切换罢了。

```
case PLUS : d = (++d>50) ? 50 : d;
case MINUS: d = (--d<1) ? 1 : d;
            break;
```

按+-键改变按方向键的增量大小。在此将增量的最大值定为 50，这个值可依个人需要自行更改。

```
case Alt_I: if(DispInfo= =ON)
            {
            putimage(0,0,buf,COPY_PUT);
            DispInfo=OFF;
            }
            else
            DispInfo=ON;
            break;
```

要决定是否显示区块信息，除了改变 DisInfo 变量外，也需注意到如果是按 Alt-I 将本功能除去的话，还必须使用 putimage 命令将原来左上角的背景数据恢复原状。

```
case Alt_K: RectXor(x, y, deltax, deltay);
            Dark(x, y, deltax, deltay);
            RectXor(x, y, deltax, deltay);
            break;
case Alt_T: RectXor(x, y, deltax, deltay);
            Light(x, y, deltax, deltay);
            RectXor(x  y, deltax, deltay);
            break;
case Alt_R: RectXor(x, y, deltax, deltay);
            Reverse(x, y, deltax, deltay);
            RectXor(x, y, deltax, deltay);
            break;
case Alt_G: free(grpdata);
            grpdata = (char *) malloc
                      (imagesize(x, y, x+deltax, y+deltay));
            RectXor(x, y, deltax, deltay);
            getimage(x, y, x+deltax, y+deltay, grpdata);
            RectXor(x, y, deltax, deltay);
            break;
case Alt_D: RectXor(x, y, deltax, deltay);
            putimage(x, y, grpdata, COPY_PUT);
            RectXor(x, y, deltax, deltay);
            break;
```

Alt-K1～Alt-R 是先将原来的方框利用 RectXor 消去后，接着调用相对应的函数做区块处理，然后再利用 RectXor 恢复原来的方框。

Alt-G 的功能是暂存区块图形。首先使用 free 函数将旧的区块图形释放，再将目前区块的图形储存到 grpdata 所指向的图形缓冲区。Alt-D 则是做相反的动作，利用 putimage 函数将已暂存的区块图形显示出来。

```
case Alt_S: {
            char filename[13];

            Getfilename(filename);
            RectXor(x, y, deltax, deltay);
            Saveimage(filename, x, y, deltax, deltay);
            RectXor(x, y, deltax, deltay);
            break;
            }

case Alt_L: {
            char filename[13];

            Getfilename(filename);
            RectXor(x, y, deltax, deltay);
            Loadimage(filename, x, y);
            RectXor(x, y, deltax, deltay);
            break;
            }
```

Alt-S 及 Alt-L 分别执行储存及载入区块图形的操作。在储存和载入之前，必须利用 Getfilename 函数输入以.TPC 结尾的字串做为文件名。

```
    case Alt_F: RectXor(x, y, deltax, deltay);
                Flipflop(x, y, deltax, deltay);
                RectXor(x, y, deltax, deltay);
                break;
    case Alt_M: RectXor(x, y, deltax, deltay);
                Mirror(x, y, deltax, deltay);
                RectXor(x, y, deltax, deltay);
                break;
    case Alt_P: RectXor(x, y, deltax, deltay);
                Hercules_printscr(x, y, x+deltax, x+deltay);
                RectXor(x, y, deltax, deltay);
                break;
}
```

Alt-F～Alt-P 分别执行上下颠倒、左右镜射及区块打印的工作。

```
RectXor(ox, oy, odeltax, odeltay);
```

```
    RectXor(x,   y,  deltax,  deltay);
    ox  =  x;
    oy  =  y;
    odeltax  =  deltax;
   odeltay  =  deltay;
```

这段程序码的主要工作为更新区块方框．首先以 RectXor(ox,oy,odeltax,odeltay)消去旧的方框，再以 RectXor(x,y,deltax,deltay)画上新的方框．最后再更新 ox，oy 及 odeltax，odeltay．

```
    if(DispInfo = = ON)      {
      char  string[30];
      sprintf(string, "X=%3d Y=%3d WX=%3d Wy=%3d", x, y, deltax, deltay);
      OutBitmapText(0,0,string,NORMAL);
    }
  }
  free(buf);
  }
```

循环的最后，判断是否要显示方框信息，如果要的话（DispInfo = = ON)，则利用 sprintf 函数将数值数据转换成字串，再利用 OutBitmapText 函数输出到屏幕的左上角．OutBitmapText 函数是一个自已编写、利用 BIOS 字形输出的模块．它的操作简单，可轻易覆盖原背景文字（绘图文字）而不会重叠，这个函数稍后将作介绍．

以上便是 MGETPIC 的主要部分．可以用如下的主函数写法进行调用：

```
void  mgetpic(void);
void OutBitmapText(int, int, char  *, int);

main()
{
int  graph__driver,  graph__mode;
int  i;
char  * buff;

detectgraph(&graph__driver,  &graph__mode);
if(  graph__driver!=HERCMONO  )  {
      puts("This  program  requires  Mono-Monitor  with  Hercules  card");
      exit(0);
}

initgraph(&graph__driver,  &graph__mode,  "");

buff=(char * )malloc(23254);
fp(fopen("g1.b1k","rb");
fread(buff,23254,1,fp);
```

```
}

*(buff+2)-=1;
putimage(0,0,buff,0);
fclose(fp);
free(buff);

mgetpic();
closegraph();
}
```

其执行结果打印如下:

(1)储存区块

Filename test tpc
影齡舊夢

```
}

*(buff+2)-=1;
putimage(0,0,buff,0);
fclose(fp);
free(buff);

mgetpic();
closegraph();
}
```

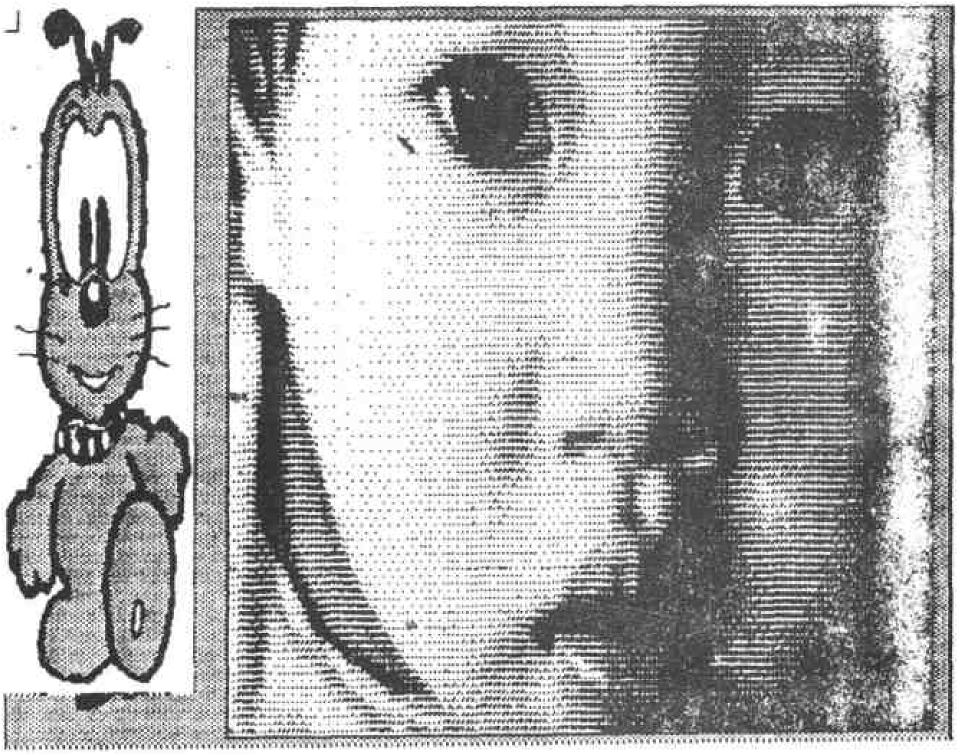
其执行结果打印如下:

(1)储存区块

(2)显示区块的相关信息

(3)区块图形镜射

(4)区块图形上下颠倒、拷贝、变暗



(5)载入 TPC 图形文件

图 6-7　MGETPIC.C 程序的执行结果

现在让我们来看看各个函数是如何完成工作的.

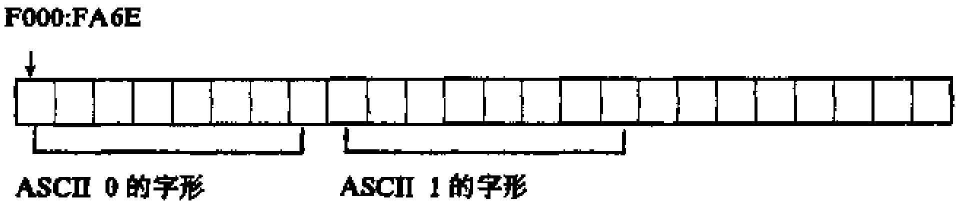
```
void RectXor(int sx, int sy, int dx, int dy)
{
    setcolor(1);
    setwritemode(XOR_PUT);
    line(sx, sy, sx+dx, sy);
    line(sx+dx, sy+1, sx+dx, sy+dy);
    line(sx+dx-1, sy+dy, sx, sy+dy);
    line(sx, sy+dy-1, sx, sy+1);
    setwritemode(COPY_PUT);
}
```

函数 RectXor 的功能是在屏幕上画上一个矩形. 由于是以 XOR 运算的方式画在屏幕上的, 所以在同一个地方同时画上两次, 就会恢复原状, 不致破坏背景. 另外请注意: 我们不直接使用 rectangle 函数的原因是以 rectangle 在 XOR 模式下绘出来的矩形四个角会因为被重复两次而消失, 所以我们用 line 函数绘出矩形, 同时避免角落的四点被重复画两次, 以维持矩形的完整性.

```
void OutBitmapText(int x, int y, char *string, int mode)
{
    register int i, j;
     unsigned addr[348];
    int len;

    for(i=0; i<348; i++)
    addr[i] = 8192 * (i%4) + 90 * (i/4);

    len = strlen(string);
    for(i=0; i<len; i++)
    for(j=0; j<8; j++)
      if(mode==NORMAL)
        pokeb(0xb000, addr[j+y]+i+x,
              peekb(0xf000, 0xfa6e+string[i] * 8+j));
      else
      pokeb(0xb000, addr[j+y]+i+x,
            peekb(0xf000, 0xfa6e+string[i] * 8+j));
}
```

本函数的功能是在 (x, y) 的坐标处显示绘图文字. 其中 x 的值是以字节为单位, 范围可介于 0 到 89 之间. 就如前几章所说明的, Hercules 适配器的图形在内存中并不呈线性排列, 所以在显示绘图文字之前, 必须先利用一个循环建立一个查表用的表格 addr, 它记载着每条屏幕扫描线在内存中的起始地址.

因为本函数所要输出的是位映射式(Bit-Map)的字形, 所以必须有每一字的字形数据可供取用. 然而这些字形数据并无需由我们准备, ROM-BIOS 本身就提供有 ASCⅡ的前 128 个字符的 8×8 字形数据, 储存于地址 F000:FA6E 的地方, 其储存格式如下:



我们只要知道字符的 ASCⅡ 值就可利用其 ASCⅡ 值乘以 8 得到它的字形数据的起始地址。只需利用 peekb 连续取出 8 个字节存入屏幕对应的内存处即可。

这个函数也提供正常显示及反白显示两种显示模式，其差别只是在输出到屏幕前，控制是否要将取得的字形数据加上反向运算罢了。

取得文件名的函数 Getfilename 较为复杂，其分析如下：

```
void Getfilename(char * filename)
{
    char * bkdata;
    int  quit=FALSE;
    int  key;
    int  CharCount=0;
    char ch;
```

定义变量。bkdata 字符指针指向储存背景图形数据缓冲区的起始地址，而 quit 为判断是否离开对话循环的变量。Char Count 记录着目前输入字数。

```
  delay(0);
  filename[0]='\0';
  bkdata = (char * ) malloc(imagesize(0,0,160,9));
  getimage(0,0,160,9,bkdata);
  OutBitmapText(0,0,"Filename:            ",REVERSE);
```

delay(0)是为了修改 Turbo C 2.0 对 delay 处理时的小错误。如果在正式用到 delay 函数之前未加上此行，则第一个 delay 函数所停留的时间会比程序所指定的还长。

第一段程序主要是初始化变量并保存背景图形。

```
  while(!quit) {
    if(kbhit()) key = bioskey(0);
    else        key = 0;
```

进入循环后先判断有没有按键，如果有则将键值储存到 key 变量，否则就令 key=0 以避免程序使用上一个键值做为判断依据。

```
switch(key) {
    case BACK : if((--CharCount)<0)
                {
                sound(1000); delay(100); nosound();
                CharCount++;
                break;
                }
```

```
                    filename[CharCount] = '\0';
                    break;
```

如果按下退位键，则检查是否所有已输入的字符都被消去，若是，则发出哗声通知用户，否则就消去一个字符（将目前字数指针 CharCount 所指到的字符存入字符结束符号'\0'）。

```
    case ENTER: quit = TRUE;
                    break;
```

按下 ENTER 键则退出返回。

```
    case SPACE: break;
```

按空格键则予以忽略。

```
      default   : ch = (char) (key & 0xff);
                    if(ch==0) break;
                    if((++CharCount)>8)
                    {
                    sound(1000); delay(100); nosound();
                    CharCount--;
                    break;
                    }
                    filename[CharCount-1] = ch;
                    filename[CharCount] = '\0';
                    OutBitmapText(9,0,"          ",REVERSE);
                    OutBitmapText(9,0,filename,REVERSE);
                    break;
    }
```

如果是其它的按键，则检查看看是否为特殊按键，key 变量的低位若为 0 即为特殊按键，则将这个字符的值加入 filename 中，并将 CharCount 加 1，重新显示一次新字串。

```
    if(key!=ENTER) {
         OutBitmapText(CharCount+9, 0, "_",REVERSE);
         OutBitmapText(CharCount+9, 0, " ",REVERSE);
    }
  }
```

利用两个 OutBitmapText 函数作光标闪烁的效果。

```
    strcat(filename, ".TPC");
    putimage(0,0,bkdata,COPY_PUT);
    free(bkdata);
  }
```

最后将标准扩展文件名.TPC 加入刚输入的字串中，恢复背景，释放内存，Getfilename 函数便告完成。

```
void Light(int sx, int sy, int dx, int dy)
{
       register int i;

       setcolor(1);
       for(i=sx; i<=sx+dx; i+=2)
       line(i, sy, i, sy+dy);
}

void Dark(int sx, int sy, int dx, int dy)
{
       register int i;

       setcolor(0);
       for(i=sx; i<=sx+dx; i+=2)
       line(i, sy, i, sy+dy);
       setcolor(1);
}
```

要使区块图形变亮或变暗最简单的方法就是将其奇数条或偶数条的垂直线（水平线亦可）全部设为亮或是不亮，即可看到此两种效果。

```
void Reverse(int sx, int sy, int dx, int dy)
{
       register int i;

       setcolor(1);
       setwritemode(XOR_PUT);
       for(i=sx; i<=sx+dx; i++)
       line(i, sy, i, sy+dy);
       setwritemode(COPY_PUT);
}
```

要区块反白显示，只须利用 XOR 逻辑运算的特性，使用直线功能在 XOR 模式下填满此区块，即可得到反白的效果。我们采用 line()而不直接使用 bar()的原因是：bar()函数并不支持 XOR 模式的绘图。

```
void Mirror(int sx, int sy, int dx, int dy)
{
     int ex, ey;
     int i, j;
     unsigned color;
     char *temp;

     ex = sx + dx;
```

```
    ey = sy + dy;
    for(i=sy; i<=ey; i++)
     for(j=sx; j<=ex; j++)    {
            setactivepage(0);
            color = getpixel(j, i);
            setactivepage(1);
            putpixel(ex-j+sx, i, color);
     }
    temp = (char *) malloc(imagesize(sx, sy, ex, ey));
    getimage(sx, sy, ex, ey, temp);
    setactivepage(0);
    putimage(sx, sy, temp, COPY_PUT);
    free(temp);
}
```

要达到 Mirror 左右镜射的效果，必须再用到另一块内存缓冲区，由于 Hercules 单色适配器附有 2 页的图形数据页（第 0 页和第 1 页），所以我们直接使用第 1 页的缓冲区。Mirror 函数的作用原理如图 6-8 所示:

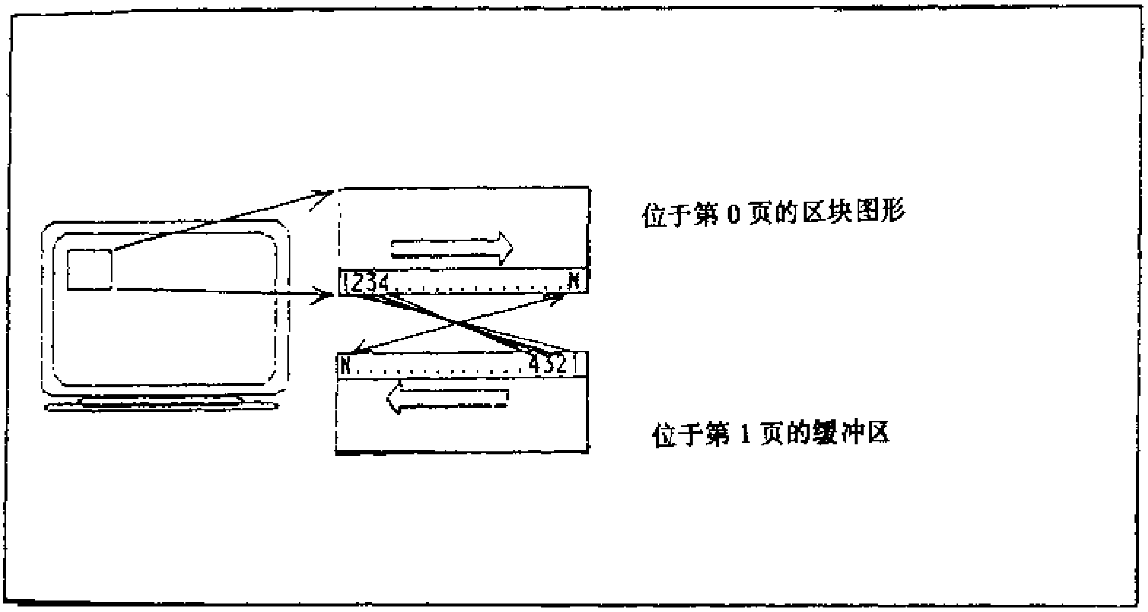


图 6-8　Mirror 函数的原理

如上图所示，程序先以每条垂直线为单位，将第 0 页的区块图形以相反的顺序存入位于第 1 页相对应的内存缓冲区，然后再将第 1 页的数据拷贝一份覆盖到原来第 0 页的区块图形上，即可完成本功能。

```
void Flipflop(int sx, int sy, int dx, int dy)
{
  int ex, ey;
  int i, j;
  unsigned color;
```

```
    char  * temp;

    ex = sx + dx;
    ey = sy + dy;
    for(i=sy; i<=ey; i++)
        for(j=sx; j<=ex; j++)    {
                setactivepage(0);
                color = getpixel(j, i);
                setactivepage(1);
                putpixel(j, ey-i+sy, color);
        }
    temp = (char  * ) malloc(imagesize(sx, sy, ex, ey));
    getimage(sx, sy, ex, ey, temp);
    setactivepage(0);
    putimage(sx, sy, temp, COPY_PUT);
    free(temp);
}
```

由于上下颠倒的原理基本上和左右镜射的原理是相同的，只不过上下颠倒功能所改变的是水平线的顺序而非垂直线的顺序。

```
void Saveimage(char  * filename, int sx, int sy, int dx, int dy)
{
      char  * buf;
      FILE  * fp;
      unsigned size;

      size = imagesize(sx, sy, sx+dx, sy+dy);
      buf = (char  * ) malloc(size);
      getimage(sx, sy, sx+dx, sy+dy, buf);
      if((fp=fopen(filename, "wb"))==NULL) {
             char  * bk;
             bk = (char  * ) malloc(imagesize(0,0,160,9));
             getimage(0,0,160,9,bk);
             sound(1000); delay(100); nosound();
             OutBitmapText(0,0,"File error writing!",REVERSE);
             getch();
             putimage(0,0,bk,COPY_PUT);
             free(bk);
             return;
      }
      fwrite(buf, size, 1, fp);
      fclose(fp);
      free(buf);
  }
```

储存图形的第一个步骤是将图形数据利用 getimage()函数存入内存缓冲区(buf)中，再使用 fwrite 函数将缓冲区内全部的数据存入文件中即可，而载入图形文件的步骤则与此相反。

```
void Loadimage(char * filename, int sx, int sy)
{
      char * buf;
      FILE * fp;
      unsigned size;

      if((fp=fopen(filename, "rb"))==NULL) {
            char * bk;
            bk = (char *) malloc(imagesize(0,0,160,9));
            getimage(0,0,160,9,bk);
            sound(1000); delay(100); nosound();
            OutBitmapText(0,0,"File error reading!",REVERSE);
            getch();
            putimage(0,0,bk,COPY_PUT);
            free(bk);
            return;
      }
      size = filelength(fileno(fp));
      buf = (char *) malloc(size);
      fread(buf, size, 1, fp);
      putimage(sx, sy, buf, COPY_PUT);
      fclose(fp);
      free(buf);
}
```

除了以上函数之外，打印的函数 Hercules_printscr 已经在 6-2 节中介绍过了，不再重述，最后是绘图、打印程序 MGETPIC.C 完整的列表：

```
/* MGETPIC.C */

#include <graphics.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include "key.h"

#define TRUE      1
#define FALSE     0
#define ON        1
```

```c
#define  OFF        0
#define  POST       0
#define  DELTA      1
#define  MaxX       720
#define  MaxY       348
#define  NORMAL     0
#define  REVERSE    1

void mgetpic(void);
void OutBitmapText(int, int, char *, int);

main()
{
    int graph_driver, graph_mode;
    int i;
    FILE *fp;
    char *fp;
    char *buff;
    buff=(char *)malloc(23254);
    detectgraph(&graph_driver, &graph_mode);
    if( graph_driver!=HERCMONO ) {
      puts("This program requires Mono-Monitor with Hercules card");
      exit(0);
    }

    initgraph(&graph_driver, &graph_mode, "");
    fp=fopen("gl.blk","rb");
    fread(buff,23254,1,fp);
    putimage(0,0,buff,0);
    fclose(fp);
    free(buff);

    mgetpic();
    closegraph();
}

void mgetpic()
{
  int x=0, y=0, ox=0, oy=0;
  int deltax=10, deltay=10, odeltax=10, odeltay=10;
  int d=1;
  int key;
  int quit=FALSE;
  int IncDecFlag=POST;
```

```
int DispInfo=OFF;
int i;
char *buf, *grpdata=NULL;
void RectXor(int, int, int, int);
void Light(int, int, int, int);
void Dark(int, int, int, int);
void Reverse(int, int, int, int);
void Getfilename(char *);
void Saveimage(char *, int, int, int, int);
void Loadimage(char *, int, int);
void Flipflop(int, int, int, int);
void Mirror(int, int, int, int);
void Hercules_printscr(int, int, int, int);

buf = (char *) malloc(imagesize(0,0,216,9));
getimage(0,0,216,9,buf);
RectXor(x, y, deltax, deltay);
while(!quit) {
 key = bioskey(0);
  switch(key) {
    case ESC  : quit = TRUE;
                break;
    case UP   : if(IncDecFlag==POST)
                {
                y -= d;
                y = (y<0) ? 0 : y;
                }
                else
                {
                deltay -= d;
                deltay = (deltay<1) ? 1 : deltay;
                }
                break;
    case DOWN : if(IncDecFlag==POST)
                {
                y += d;
                y = ((y+deltay)>=MaxY) ? y-d : y;
                }
                else
                {
                deltay += d;
                deltay = ((deltay+y)>=MaxY) ? deltay-d : deltay;
                }
                break;
```

```
case RIGHT: if(IncDecFlag==POST)
            {
            x += d;
            x = ((x+deltax)>=MaxX) ? x-d : x;
            }
            else
            {
            deltax += d;
            deltax = ((deltax+x)>=MaxX) ? deltax-d : deltax;
            }
            break;
case LEFT : if(IncDecFlag==POST)
            {
            x -= d;
            x = (x<0) ? 0 : x;
            }
            else
            {
            deltax -= d;
            deltax = (deltax<1) ? 1 : deltax;
            }
            break;
case SPACE: IncDecFlag = (IncDecFlag==POST) ? DELTA : POST;
            break;
case PLUS : d = (++d>50) ? 50 : d;
            break;
case MINUS: d = (--d<1) ? 1 : d;
            break;
case Alt_I: if(DispInfo==ON)
            {
            putimage(0,0,buf,COPY_PUT);
            DispInfo=OFF;
            }
            else
            DispInfo=ON;
            break;
case Alt_K: RectXor(x, y, deltax, deltay);
            Dark(x, y, deltax, deltay);
            RectXor(x, y, deltax, deltay);
            break;
case Alt_T: RectXor(x, y, deltax, deltay);
            Light(x, y, deltax, deltay);
            RectXor(x, y, deltax, deltay);
            break;
```

```
case Alt_R: RectXor(x, y, deltax, deltay);
            Reverse(x, y, deltax, deltay);
            RectXor(x, y, deltax, deltay);
            break;
case Alt_G: free(grpdata);
            grpdata = (char * ) malloc
                      (imagesize(x, y, x+deltax, y+deltay));
            RectXor(x, y, deltax, deltay);
            getimage(x, y, x+deltax, y+deltay, grpdata);
            RectXor(x, y, deltax, deltay);
            break;
case Alt_D: RectXor(x, y, deltax, deltay);
            putimage(x, y, grpdata, COPY_PUT);
            RectXor(x, y, deltax, deltay);
            break;
case Alt_S: {
            char filename[13];

            Getfilename(filename);
            RectXor(x, y, deltax, deltay);
            Saveimage(filename, x, y, deltax, deltay);
            RectXor(x, y, deltax, deltay);
            break;
            }
case Alt_L: {
            char filename[13];

            Getfilename(filename);
            RectXor(x, y, deltax, deltay);
            Loadimage(filename, x, y);
            RectXor(x, y, deltax, deltay);
            break;
            }
case Alt_F: RectXor(x, y, deltax, deltay);
            Flipflop(x, y, deltax, deltay);
            RectXor(x, y, deltax, deltay);
            break;
case Alt_M: RectXor(x, y, deltax, deltay);
            Mirror(x, y, deltax, deltay);
            RectXor(x, y, deltax, deltay);
            break;
case Alt_P: RectXor(x, y, deltax, deltay);
            Hercules_printscr(x, y, x+deltax, x+deltay);
            RectXor(x, y, deltax, deltay);
```

```
                    break;
        }
        RectXor(ox, oy, odeltax, odeltay);
        RectXor(x,   y,  deltax,  deltay);
        ox = x;
        oy = y;
        odeltax = deltax;
        odeltay = deltay;
        if(DispInfo==ON)    {
          char string[30];
          sprintf(string, "X=%3d Y=%3d WX=%3d Wy=%3d", x, y, deltax, deltay);
          OutBitmapText(0,0,string,NORMAL);
        }
    }
    free(buf);
    }

void RectXor(int sx, int sy, int dx, int dy)
{
  setcolor(1);
  setwritemode(XOR_PUT);
  line(sx, sy, sx+dx, sy);
  line(sx+dx, sy+1, sx+dx, sy+dy);
  line(sx+dx-1, sy+dy, sx, sy+dy);
  line(sx, sy+dy-1, sx, sy+1);
  setwritemode(COPY_PUT);
}

void OutBitmapText(int x, int y, char *string, int mode)
{
  register int i, j;
  unsigned addr[348];
  int len;

  for(i=0; i<348; i++)
    addr[i] = 8192 * (i%4) + 90 * (i/4);

  len = strlen(string);
  for(i=0; i<len; i++)
    for(j=0; j<8; j++)
      if(mode==NORMAL)
        pokeb(0xb000, addr[j+y]+i+x,
              peekb(0xf000, 0xfa6e+string[i]*8+j));
      else
```

```
        pokeb(0xb000, addr[j+y]+i+x,
              peekb(0xf000, 0xfa6e+string[i] * 8+j));
}

void Reverse(int sx, int sy, int dx, int dy)
{
        register int i;

        setcolor(1);
        setwritemode(XOR_PUT);
        for(i=sx; i<=sx+dx; i++)
        line(i, sy, i, sy+dy);
        setwritemode(COPY_PUT);
}

void Light(int sx, int sy, int dx, int dy)
{
        register int i;

        setcolor(1);
        for(i=sx; i<=sx+dx; i+=2)
        line(i, sy, i, sy+dy);
}

void Dark(int sx, int sy, int dx, int dy)
{
        register int i;

        setcolor(0);
        for(i=sx; i<=sx+dx; i+=2)
        line(i, sy, i, sy+dy);
        setcolor(1);
}
void Getfilename(char *filename)
{
   char *bkdata;
   int  quit=FALSE;
   int  key;
   int  CharCount=0;
   char ch;

   delay(0);
   filename[0]='\0';
   bkdata = (char *) malloc(imagesize(0,0,160,9));
```

```
    getimage(0,0,160,9,bkdata);
    OutBitmapText(0,0,"Filename:              ",REVERSE);
    while(!quit) {
      if(kbhit()) key = bioskey(0);
      else        key = 0;
      switch(key) {
        case BACK : if((--CharCount)<0)
                    {
                    sound(1000); delay(100); nosound();
                    CharCount++;
                    break;
                    }
                    filename[CharCount] = '\0';
                    break;
       case ENTER: quit = TRUE;
                    break;
       case SPACE: break;
       default    : ch = (char) (key & 0xff);
                    if(ch==0) break;
                    if((++CharCount)>8)
                    {
                    sound(1000); delay(100); nosound();
                    CharCount--;
                    break;
                    }
                    filename[CharCount-1] = ch;
                    filename[CharCount] = '\0';
                    OutBitmapText(9,0,"        ",REVERSE);
                    OutBitmapText(9,0,filename,REVERSE);
                    break;
    }
    if(key!=ENTER) {
         OutBitmapText(CharCount+9, 0, "_",REVERSE);
         OutBitmapText(CharCount+9, 0, " ",REVERSE);
    }
  }
  strcat(filename, ".TPC");
  putimage(0,0,bkdata,COPY_PUT);
  free(bkdata);
  }

void Saveimage(char *filename, int sx, int sy, int dx, int dy)
{
      char *buf;
```

```
        FILE  *fp;
        unsigned size;

        size = imagesize(sx, sy, sx+dx, sy+dy);
        buf = (char *) malloc(size);
        getimage(sx, sy, sx+dx, sy+dy, buf);
        if((fp=fopen(filename, "wb"))==NULL) {
                char *bk;
                bk = (char *) malloc(imagesize(0,0,160,9));
                getimage(0,0,160,9,bk);
                sound(1000); delay(100); nosound();
                OutBitmapText(0,0,"File error writing!",REVERSE);
                getch();
                putimage(0,0,bk,COPY_PUT);
                free(bk);
                return;
        }
        fwrite(buf, size, 1, fp);
        fclose(fp);
        free(buf);
}

void Loadimage(char *filename, int sx, int sy)
{
        char *buf;
        FILE  *fp;
        unsigned size;

        if((fp=fopen(filename, "rb"))==NULL) {
                char *bk;
                bk = (char *) malloc(imagesize(0,0,160,9));
                getimage(0,0,160,9,bk);
                sound(1000); delay(100); nosound();
                OutBitmapText(0,0,"File error reading!",REVERSE);
                getch();
                putimage(0,0,bk,COPY_PUT);
                free(bk);
                return;
        }
        size = filelength(fileno(fp));
        buf = (char *) malloc(size);
        fread(buf, size, 1, fp);
        putimage(sx, sy, buf, COPY_PUT);
        fclose(fp);
```

```
        free(buf);
    }

    void Flipflop(int sx, int sy, int dx, int dy)
    {
        int ex, ey;
        int i, j;
        unsigned color;
        char * temp;

        ex = sx + dx;
        ey = sy + dy;
        for(i=sy; i<=ey; i++)
            for(j=sx; j<=ex; j++)    {
                setactivepage(0);
                color = getpixel(j, i);
                setactivepage(1);
                putpixel(j, ey-i+sy, color);
            }
        temp = (char *) malloc(imagesize(sx, sy, ex, ey));
        getimage(sx, sy, ex, ey, temp);
        setactivepage(0);
        putimage(sx, sy, temp, COPY_PUT);
        free(temp);
    }

    void Mirror(int sx, int sy, int dx, int dy)
    {
        int ex, ey;
        int i, j;
        unsigned color;
        char * temp;

        ex = sx + dx;
        ey = sy + dy;
        for(i=sy; i<=ey; i++)
            for(j=sx; j<=ex; j++)    {
                setactivepage(0);
                color = getpixel(j, i);
                setactivepage(1);
                putpixel(ex-j+sx, i, color);
            }
        temp = (char *) malloc(imagesize(sx, sy, ex, ey));
        getimage(sx, sy, ex, ey, temp);
```


17

```
        setactivepage(0);
        putimage(sx, sy, temp, COPY_PUT);
        free(temp);
  }
  void Hercules_printscr(int sx, int sy, int ex, int ey)
  {
    int deltax;
    int n1, n2;
    int i, j, k;
    char data;
    void p_putc(char);
    deltax = ex - sx + 1;
    p_putc(27);
    p_putc('3');
    p_putc(20);
    n1 = deltax % 256;
    n2 = deltax / 256;
    for(i=sy; i<=ey; i+=8)  {
        p_putc(27);
        p_putc('L');
        p_putc(n1);
        p_putc(n2);
        for(j=sx; j<=ex; j++)          {
              data = 0;
              for(k=0; k<8; k++)
              if(getpixel(j, i+k))
                  data |= (1<<(7-k));
              p_putc(data);
        }
        p_putc('\n');
        p_putc('\r');
 }
   p_putc(27);
   p_putc('2');
 }
 void p_putc(char ch)
 {
       fprintf(stdprn, "%c", ch);
 }
```

第七章　驻留式绘图程序设计实例

7.1　驻留程序简介

在本章中，我们以第六章的绘图与图形打印程序为基础，将它改写成驻留式的绘图程序并将它命名为“绘图小精灵”．我们的绘图小精灵和目前市面上的几种绘图程序比较起来孰优孰劣呢？就请读者自行判断吧！

一般的程序在运行完毕后，会从主存中消除，归还所有向操作系统取得的资源，以供后来的程序使用，而驻留式却不然．驻留程序是一种结束仍驻留(Terminate and Stay Resident,TSR)的程序．这种类型的程序在运行之后，就将本身的大部分（或全部）的程序码及数据驻留在内存中，即使程序运行结束后，留在主存中的程序码及数据仍然受到操作系统的保护，不被后来运行的程序占用．利用这一些特殊方法，这个程序可以立刻再度取得控制权，运行用户所要运行的任务而不须再由磁盘中载入重新运行，达到随叫随到的特性．有了这些特性，我们可以做到许多一般正常程序所无法达到的功能．

现在我们可以由驻留程序设计一个简单的小闹钟，随时提醒我们现在的时刻．设计一个计算器让我们在任何环境下计算数值或转换各种进制，或者设计一个绘图程序以截取喜欢的图形、打印程序打印全页画面或部分画面……．当然本书并非讨论驻留程序的专书，限于篇幅，上述软件的程序设计技巧无法在此一一详述．但是我们仍将使用 C 语言来写驻留程序的原理及注意事项作尽量详细的解说，让读者能充分发挥驻留程序的威力，设计出实用的驻留程序．

由于驻留程序运行后会将本身驻留于内存中，因此对于程序的大小相当苛刻．这一点，利用 C 语言来编写驻留程序并非最佳的选择，但是顾及程序的可读性及易于说明，初学者使用 C 语言写驻留程序，也不失为一个好的方法．反正原理大致相同，假若你已经熟悉了写驻留程序的方法，并且懂得汇编语言，要将你的 C 程序转成汇编语言并非难事．

以高级语言编写驻留程序除了多占内存空间之外，还有一项很严重的缺点，就是我们无法掌握程序的运行．在高级语言中，为了用户的方便，提供了许多相当好的函数，但是我们并没有源代码，无法得知这些函数到底用什么方法达到它的目的．例如在函数中有没有用到会和驻留程序相冲突的功能调用，使得我们在使用函数之前便必须先造成一个彻底安全的环境，或者避免用到确实危险（会造成随机）的函数．

7.2　Turbo C 编写驻留程序的方法

7.2.1　keep 函数

撰写驻留程序所面临的第一个问题当然就是如何使程序驻留于主存中．在 Turbo C 中有一个函数 keep 便具有此功能，其格式及语法如下：

定义：#include <dos.h>

void keep(unsigned char code ,unsigned size);

调用：keep(0,100);

keep(0,(FP_G(_heapbase)-_sp(/ 16+1);

说明：code:返回值

size:程序保留在内存中的大小，以段(Parggraph)为单位

这个函数实际上是对 DOS 0x31 号的功能调用，你只要指定要驻留的内存大小，并以段（每 16 个字节）为计数单位告诉 keep 函数，它便自动将你的程序由程序头算起，到你指定的大小处驻留于内存中，然后将返回值返回给父程序（通常是 COMMAND.COM），接着结束程序。那么如何计算要驻留程序的大小呢？主要有以下两个方法：

1. 使用命令编译方式加上-M 选项：

C:\TC>tcc-M-ls tick

如此 Turbo C 的编译程序便会产生该程序的 MAP 文件表：

Start	Stop	Length	Name	Class
00000H	00745H	00746H	_TEXT	CODE
00750H	008F7H	001A8H	_DATA	DATA
008F8H	008FBH	00004H	_EMUSEG	DATA
008FCH	008FDH	00002H	_CRTSEG	DATA
008FEH	008FEH	00000H	_CVTSEG	DATA
008FEH	008FEH	00000H	_SCNSEG	DATA
008EFH	00953H	00056H	_BSS	BSS
00954H	00954H	00080H	_BSSEND	STACK
00960H	009DFH	00080H	_STACK	STACK

接着观察 MAP 文件前端的各节区长度记录，将所有节区的长度相加，再加上程序中使用_heaplen 变量所指定的堆栈长度（缺省值为 64K），及使用_stklen 变量所指定的堆栈长度（缺省值 4K）即得到需驻留的程序长度。使用这个方法可以有效地节省内存的使用，但是却较难以掌握，除非你已经对 Turbo C 相当熟悉了，否则必须经过多次尝试，再加上一个安全的值以得到较稳定的驻留程序。

2. 由于Turbo C在程序存储分配方面有其固定的格式，以小型存储模式为例，它的分配方式就像下面这个样子：

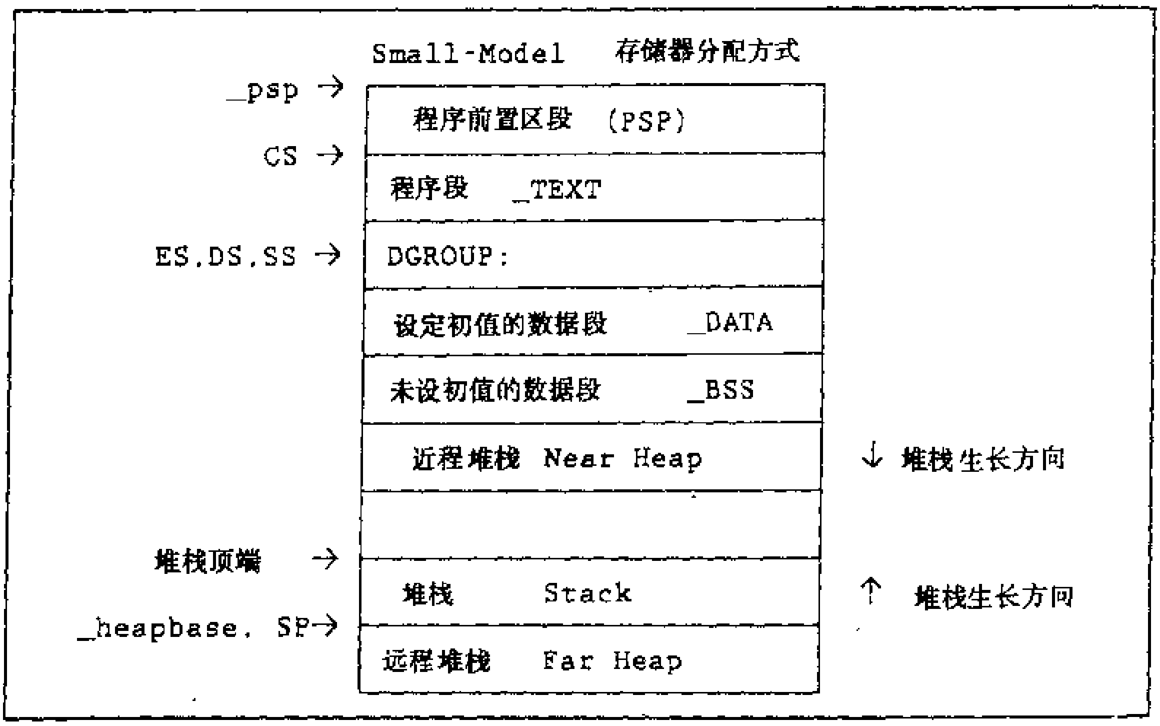


图 7-1 Turbo C 程序的小规模存储分配情形

在图 7-1 中可以发现，其前后端各有一变量指向，因此我们可以用下列的式子驻留我们的程序，本书也将以这种方式来编写以后的驻留程序：

keep(0,FP__EG(__eapbase)-__psp);

7.2.2 运行驻留程序的方法

程序驻留之后，如果没有运行的话就失去驻留的意义了. 一般而言，驻留程序依其被运行的方式，可分为主动型和被动型两种：

1. 主动型的驻留程序可由用户接着按某一特定的按键组合来调用，而这个按键组合就是所谓的热键.
2. 被动型的驻留程序通常都是不可直接由用户调用的. 在此，我们将着眼于主动型的驻留程序.

驻留程序驻留在内存后，必须随时检测用户是否按了热键，以判断是否要运行驻留程序. 如何检测呢？首先我们必须先了解当用户按下了按键之后，PC 是如何处理的.

当用户按下键盘的时候，键盘会产生一个硬件的 9 号中断将该键的扫描码(scan code)及 ASCⅡ码存入 BIOS 数据区 0040:001E 到 0040:003D 的键盘缓冲区. 如果用户按下的是 shift，Alt，Ctrl……等特殊的按键，也会同时在 BIOS 数据区中的 0040:0017 和 0040:0018 两个字节中记录下这些特殊键的情形.

由于驻留程序通常都在各种不同的程序后台检查热键，为了避免热键的设置和前台程序的某些按键汇编相同，以致于影响到前台程序的正常运行，常驻程序的按键都尽量选用较特殊的按键汇编. 正因此，这两个字节对我们而言也就特别重要，现将此两位字节各位

所代表的意义分别说明表 7–1:

表 7–1　DOS 的堆栈使用原理

住址　0040:0017

D7	D6	D5	D4	D3	D2	D1	D0

D0＝1　:　右边Shift键按下
D1＝1　:　左边Shift键按下
D2＝1　:　Ctrl键按下
D3＝1　:　Ctrl键按下
D4＝1　:　ScrollLock打开
D5＝1　:　NumLock打开
D6＝1　:　CapsLock打开
D7＝1　:　Insert打开

地址　0040:0017

D7	D6	D5	D4	D3	D2	D1	D0

D0　:　未使用
D1　:　未使用
D2　:　未使用
D3＝1　:　Ctrl–Number按下
D4＝1　:　Scroll Lock按下
D5＝1　:　NumLock按下
D6＝1　:　CapsLock按下
D7＝1　:　Insert按下

检测按键除了检查 0040:0017 及 0040:0018 来判别特殊按键外，我们也可自行设计一段程序串接在 INT9 后面（亦即自行写一个包含旧的 INT9 在内的新 INT9），只要有任何按键的动作，马上跟着运行我们的检测热键的任务，新的 INT9 应该像下图 7–2 所示，

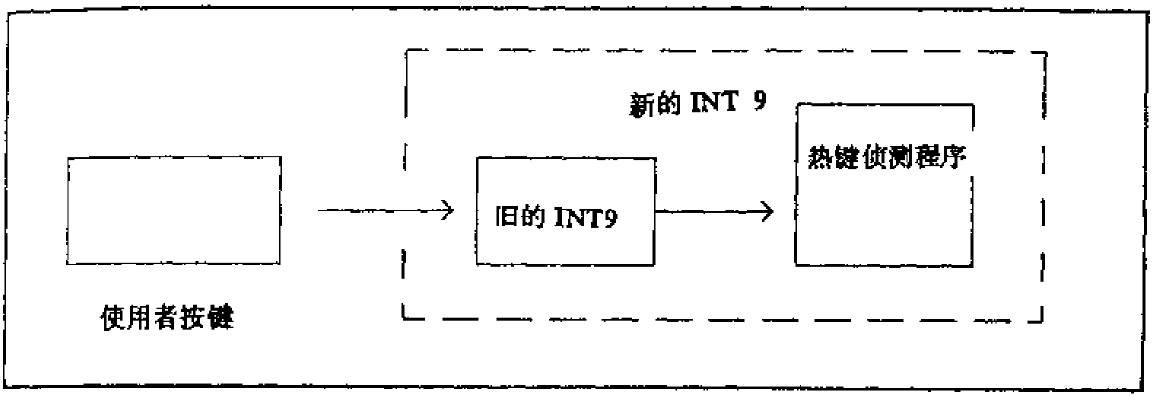


图 7–2　TSR 的键盘处理

但是假如用户按下的是 PrintScreen 键、Alt-Ctrl-Del 键及 Ctrl-Break 等组合键，那么 INT9 中断服务子程序并不会正常地结束，而是立刻跳到相对应的 INT5，INT9 及 INT18 等中断去运行相对应的功能。所以如果想拦截这三种组合键，一定要把检查的程序挂在 INT9 之前，先行读键检查。这个技术我们会在程序部分详细讨论。

驻留程序的另一个取得控制权的方法是利用计时器中断 1ch。PC 上有一个系统时钟信号发生器，每秒产生 18.2 个脉冲激活系统计时器中断 INT08，而 INT08 的中断服务子程序中也会调用一次软件中断 INT 1ch；亦即 CPU 每$\frac{1}{18.2}$秒便会来运行一次 INT 1ch。由于 INT 1ch 本身是一个只有一道 iret 指令的中断，我们便可以轻易地将 INT 1ch 的中断矢量指向我们的 TSR 程序，这样我们的 TSR 程序便每$\frac{1}{18.2}$秒就可以取得一次控制权。

7.3 简单的驻留程序示例

7.3.1 键盘发声程序

下面我们先利用 INT 9 及 INT 1ch 分别写一个驻留程序，接着讨论此驻留程序，再分析驻留程序所会遇到的问题。首先是一个修改 INT 9 的键盘发声程序 NOISE.C；这个程序将一小段发声的程序附加在原来的 INT 9 后面，使得每次用户按下键盘的任意键时，PC 的扬声器便会发出清脆的声音。由于键盘在按下及放开键时都会调用到 INT 9，所以按一下键再放开，应该会听到 2 个声音。下面就是 NOISE.C 的程序清单：

```
/ * NOISE.C * /

#include <stdio.h>
#include <dos.h>
#include <bios.h>

void interrupt new09();
void interrupt ( * old09)();

extern void far * _heapbase;

main()
{
        delay(0);
        disable();
        old09 = getvect(0x09);
        setvect(0x09, new09);
        enable();
        puts("[Noise] install OK!\n");
        keep(0, FP_SEG(_heapbase)-_psp);
        puts("Can't install TSR program!");
```

```
}

void interrupt new09()
{
	(*old09)();

	sound(1200);
	delay(5);
	nosound();
}
```

这个程序用到了 getvect()函数及 setvect()函数，分别说明如下：

定义：#include<dos.h>
　　　void setvect(int intno,void interrupt (*isr)());
调用：void interrupt isr(void);
　　　setvect(intno,isr);
说明：intno：中断号码
　　　isr　：中断服务子程序起始地址

定义：#include<dos.h>
　　　void interrupt (*getvect(int intno))();
调用：void interrupt (*isr)();
　　　isr=getvect(intno);
说明：intno：中断号码
　　　isr　：中断服务子程序起始地址

setvect()函数可以将我们程序中的任一函数指定给某个中断，而 getvect()函数则取得某个中断服务子程序的起始地址。若将 getvect()函数所取得中断服务子程序的起始地址，指定给我们已定义过的中断向量表，那么我们便可以在程序中自由地调用中断了。NOISE.C 就是利用这个函数达到连接 INT 9 及发声的目的。在程序中，首先取得原先的 INT 9 中断，将它指定给 old90()函数，然后令 INT 9 的中断矢量指向 new09()函数。如此，每当用户按下按键时，系统便会调用 new09()，再由 new09()去调用原来的 INT 9 (即 old09())，接着运行发声程序然后返回。值得注意的是：由于 new09()已被视为一个中断服务子程序，有别于一般的函数，所以必须在 new09()之前加上 interrupt 说明，子程序方能正确运行。

这个程序并没有加入判断是否重复载入的功能，所以如果你在 DOS 下多次运行本程序，那么 new09()函数将会被一直地串联下去，也就是说：你按一次键，听到的声音恐怕就不只两声了。

7.3.2 计时器驻留程序

下而这个程序是是使用计时器中断 11h 的简单驻留程序示例。运行本程序后，约每隔一秒就会发出一个清脆的声音，直到你开机重新引导计算机为止，由于 TICK.C 的原

理和 NOISE.C 相似，在此不多加说明。

```
/ * TICK.C * /

#include <stdio.h>
#include <dos.h>
#include <bios.h>

void interrupt new1c();
void interrupt (* old1c)();

extern void far *_heapbase;
static unsigned count=0;

main()
{
        delay(0);
        disable();
        old1c = getvect(0x1c);
        setvect(0x1c, new1c);
        enable();
        puts("[Tick] install OK!\n");
        keep(0, FP_SEG(_heapbase)-_psp);
        puts("Can't install TSR program!");
}

void interrupt new1c()
{
        (* old1c)();
        if(++count==18) {
              sound(1200);
              delay(5);
              nosound();
              count = 0;
        }
}
```

7.4 加强驻留程序的功能

7.4.1 驻留程序的堆栈及相关参量切换

上列两个程序因为程序都很小，所以只要驻留完整（即 keep( )函数中的 size 参量够大），运行起来都不会有问题，但是以这种简陋的写法，当程序再稍微大些时，恐怕就响声四起了。

原来 DOS 是一个单任务的操作系统，在同一时间只能管理一个程序的运行，而我们的驻留程序却会形成同时有许多程序一起在内存中运行的情况。举个例子，假设我们已经在内存中驻留了一个驻留程序 T，而后又在 DOS 的提示符号下运行一般的程序 F。就在程序 F 正在运行的时候，我们利用热键引导了驻留程序 T；此时虽然驻留程序 T 已经取得了控制权，然而 DOS 却不知道程序已经做了切换，仍然以为现在运行的程序还是 F，一些与程序有关的参量、寄存器仍然指向程序 F。因此，当驻留程序 T 取得控制权后立刻要切换这些参量，以免沿用前台程序 F 的数据及环境，造成不正确的结果。这些参量主要包括 PSP，DTA 及堆栈寄存器 SS，SP。堆栈的切换可以直接使用 Turbo C 的虚拟变量_ss 及_sp 完成，而切换 PSP 也有 Turbo C 的全局变量_psp 可用，至于 DTA 则必须运用 DOS 的功能调用 2F 及 1A 完成。

7.4.2 防止驻留程序的再进入

由于驻留程序的设计通常是只要检测到热键就立刻引导，因此假如程序本身的动作尚未做完而用户又按下了热键的话，将会再度引导驻留程序。前一个程序尚未完成，而后一个程序再次运行，所使用的是同一块数据区及堆栈区，如此原驻留程序的数据很容易遭到破坏以致无法正常运行。为了避免这种情形的发生，通常都会设计一个标志：

1. 在进人驻留程序之后立即将此标志设为 1。

2. 离开后再将标志恢复为 0。

当用户激活此驻留程序时，便先检查这个标志的内容，如果是 1 的话表示驻留程序已经在运行了，无须再进人，可立即返回；若为 0 的话，表示目前并未激活，可安心运行。

上述这个情形亦适用于时钟中断 1ch 的驻留程序。这种驻留程序每 1／18.2 秒便会自动激活一次，如果在激活的间隔时间内无法完成任务的话就会造成驻留程序的重复进人，这时，利用设置进人标志的方式即可解决这个问题。

对 CPU 而言，一个中断产生后，在运行该中断之前需先将下一个要运行的指令的地址（即返回地址）保存在一个特定的地址（通常是堆栈）上，等中断运行完毕后再取回该地址继续运行原来的程序。一般而言，驻留程序可能在任何时间、任何情况下中断正在运行的前台程序，中断的那一瞬间或许正在运行一个运算，也可能正在运行 BIOS 或 DOS 的中断服务子程序。这时，我们在驻留程序中若要使用到中断服务子程序，就必须有所选择了。

一般情况下，如果要使用 BIOS 的功能调用，由于 BIOS 本身设计有一个中断键来保存每一次中断的返回地址及标志，所以并不会产生不良的后果，可以在驻留程序中放心地使用 BIOS 大部分的功能调用。但是如果是要在驻留程序中使用到 DOS 的功能调用就要小心了。因为 DOS 在设计的时候，并没有完全考虑到功能调用重复进人的问题。以图 7–3 为例。

前台程序 F 在运行时调用了 DOS 的功能调用 A，正在运行的同时激活了驻留程序 T，假如在驻留程序 T 中又调用了 DOS 的功能调用 B 的话，此时若 A 与 B 所使用的是同一堆栈的话，那么 B 将会破坏原先 A 所创建和使用的数据，可能就会造成死机或其它不可预期的错误。因此在调用 DOS 的功能调用之前必须能够确定不要调用到使用相同堆栈的功能调用。

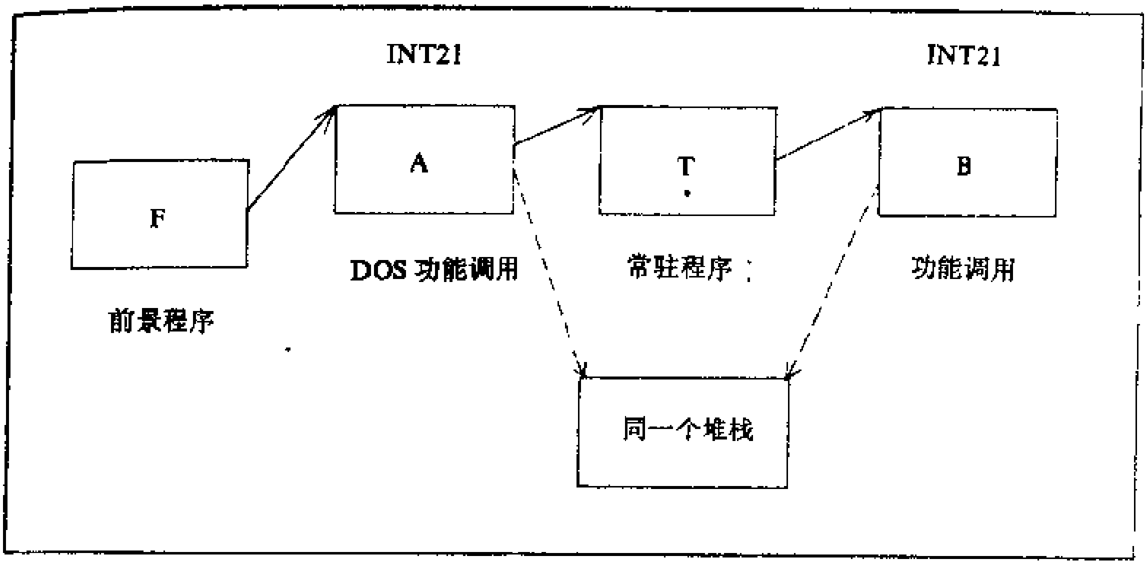


图 7-3 功能调用重复进入的问题

DOS 为功能调用准备了三个堆栈。第一个堆栈（又称为磁盘堆栈）供 00 及 0D 以上(含 0Dh)的功能调用及 INT 25h，INT 26h，INT 28h 等中断使用；第二个堆栈（又称为 I／O 堆栈）供 01～0C 等功能调用使用；第三个堆栈（又称辅助堆栈）供 59h 功能调用使用。同时假如发生了严重错误，则用来取代第二个堆栈（I／O 堆栈）。我们将 DOS 的堆栈使用情况整理如表 7-2:

表 7-2 DOS 的堆栈使用情况

功能调用／中断	使用的堆栈
INT 21h : 00h	DISK STACK
INT 21h : 01h-1Ch	ErrFlag= 0 : I／O STACK ErrFlag = 1 : AUX STACK
INT 21h : 0Dh	DISK STACK
INT 21h : 50h,51h ,62h	调用者程序的堆栈
INT 21h : 59h	AUX STACK
INT 21h : 33h	调用者程序的堆栈
INT 25h :	DISK STACK
INT 26h :	DISK STACK
INT 28h :	DISK STACK

* ErrFlag ：严重错误标志
* AUX STACK ：辅助堆栈
* DISK STACK ：磁盘堆栈

从以上的堆栈的使用情况图可以看出，只要避免使用到相同的堆栈，就可以放心地在驻留程序中作用到 DOS 的功能调用了。那我们如何得知目前前台正在使用哪一个堆栈呢？有下列几个解决方法：

7.4.2.1 观察 DOS 的忙碌标志

DOS 有一个未公开的 34h 功能调用，它的功能在返回 DOS 内部一忙碌标志的地址，我们只须在驻留程序设置之前先取得这个地址，然后每次激活驻留程序时就可以先检查此值的内容．如果是 0，表示目前并未运行 DOS 的任一个功能调用，程序中可放心使用任何一个功能调用而不会有不良的后果．

表 7-3 DOS 忙碌标志地址

功 能	:	取得DOS忙碌的标志地址
输入值	:	AH＝34H
返回值	:	ES:BX＝忙碌标志地址

7.4.2.2 为原来的 INT 21 加上判别程序

我们可以加一小段程序在 INT 21 之前，先检查每一个调用 INT 21 的 AH 之值，然后判别使用的是哪一个堆栈，将它记录于一个全局变量里，当驻留程序被激活时便可以检查这个值以作为参考．一般而言，如果我们的驻留程序中非使用到 DOS 的磁盘堆栈不可，例如必须执行文件相关功能．而此时的前台程序又在使用磁盘的话，当然我们必须避免去激活我们的驻留程序．

当 DOS 在运行 01～0C 的功能调用时，每隔 1／18.2 秒便会激活一次 INT 28，而在 DOS 初始时，INT 28 只有一行 iret 的指令，亦即空的中断．我们可以用 debug 来观察：

```
C>debug
-a100
0BB2:0100 int 28
0BB2:0102 int 20
0BB2:0104
-t

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFE8 BP=0000 SI=0000 DI=0000
DS=0BB2 ES=0BB2 SS=0BB2 CS=0116 IP=10DA NV UP DI PL NZ NA PO NC
0116:10DA CF
-t

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFE8 BP=0000 SI=0000 DI=0000
DS=0BB2 ES=0BB2 SS=0BB2 CS=0116 IP=10DA NV UP DI PL NZ NA PO NC
0BB2:0102 CD20          INT    20
-q
```

但是当我们运行 DOS 下的 PRINT.EXE 这个驻留程序后发现，INT 28 不再是空的中断，它已经被改为指向一个新的服务子程序了．

```
C>print
Name of list device [PRN] :
Resident part of PRINT installed
PRINTER queue in enpty

C>debug
-a100
0D1C:0100 int 28
0D1C:0102 int 20
0D1C:0104
-t

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFE8 BP=0000 SI=0000 DI=0000
DS=0D1C ES=0D1C SS=0D1C CS=0675 IP=0615  NV UP DI PL NZ NA PO NC
0675:0615  E856FF        CALL   056E
-q

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFE6 BP=0000 SI=0000 DI=0000
DS=0D1C ES=0D1C SS=0D1C CS=0675 IP=0615  NV UP DI PL NZ NA PO NC
0675:056E  F8         CLC
-q
```

由此可见，INT 28 其实是 DOS 预留下来写驻留程序之用的中断，如果改写这个中断，可以放心使用 DOS 中 01～0C 以外的任何一个功能调用：

7.4.2.3 更改 DOS 的严重错误标志

各位还记不记得前面曾经提过的严重错误标志 Critical-Error Flag? 原先 DOS 缺省在运行 01～0C 的功能调用时使用的是 I/O 堆栈，但是如果发生严重错误(Critical-Error)时，便会改成使用辅助堆栈。我们可以利用功能调用中的功能 50h 的子功能 06 来取得严重错误标志的地址，从这个地址，便可以在使用 01～0C 的功能调用前先将此标志设为 1，让 DOS 以为发生了严重错误而使用辅助堆栈，等到使用完毕再将此标志还原即可，这样可以让我们更方便地运用 DOS。

表 7-4 DOS 严重错误标志地址

功 能	：取得DOS严重错误标志地址
输入值	：AH=5DH
	AL=06H
返回值	：DS:SI=严重错误标志地址

但是如果只有修改严重错误标志，却将发生另外一个问题——DOS 会以为真的发生了严重错误。当 DOS 认为已经发生了严重错误时，除了切换堆栈外，它还调用 INT 24h 的严重错误处理子程序，出现出错信息，这是我们所不乐意见到的。所以当我们要在驻留

程序中应用到这个标志时，就必须修改 INT 24h 以避免这种情形发生。最简单的方法是将 INT 24h 改成什么事也不做的子程序即可。请看下面这个程序清单（INT24-1.C）：

```
void interrupt NewInt24()
{
    _AX=0;
}
```

这个新的 INT 24h 将寄存器 AX(action code)归 0，令 DOS 忽略这个错误同时什么事也不做便返回了。但是如此写却行不通！我们将这个程序编译成汇编语言看看：

```
C>tcc -S int 24-1
Turbo C Version 2.0 Copyright (c) 1987,1988 Borland Internationl
int24-1.c:
      Available memory 439696
```

以下是 INT24-1.ASM 的清单：

```
            ifndef   ? ? version
            ?debug   macro
            endm
            endif
            ?debug   S "int24-1.c"
_TEXT       segment  byte public 'CODE'
DGROUP      group    _DATA,_BSS
            assume   cs:_TEXT,ds:DGROUP,ss:DGROUP
_TEXT       ends
_DATA       segment  word public 'DATA'
d@          label    byte
d@w         label    word
_DATA       ends
_BSS        segment  word public 'BSS'
b@          label    byte
b@w         label    word
            ?debug   C E9EB055A1809696E7432342D312E63
_BSS        ends
_TEXT       segment  byte public 'CODE'

;           ?debug   L 1
_NewInt24            proc    far
            push     ax
            push     bx
            push     cx
            push     dx
            push     es
            push     ds
```

```
                push    si
                push    di
                push    bp
                mov     bp,DGROUP
                mov     ds,bp
;               ?debug  L 3
                xor     ax,ax
@1:
;               ?debug  L 4
                pop     bp
                pop     di
                pop     si
                pop     ds
                pop     es
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                iret
_NewInt24               endp
_TEXT           ends
                ?debug  C E9
_DATA           segment word public 'DATA'
s@              label   byte
_DATA           ends
_TEXT           segment byte public 'CODE'
_TEXT           ends
                public  _NewInt24
                end
```

观察了 INT24-1.ASM 之后，就会发现原来 Turbo C 遇到了 interrupt 说明的函数时，会自动地在它前面把所有寄存器堆入堆栈及退栈，使得我们在函数中直接使用虚拟变量_AX 更改寄存器 AX 的值，离开函数后又恢复原状，所以我们必须使用另外一种写法，请看程序 INT24-2.C:

```
void interrupt NewInt24(BP, DI, SI, DS, ES, DX, CX, BX, AX)
unsigned BP, DI, SI, DS, ES, DX, CX, BX, AX;
{
     AX=0;
}
```

现在将 INT24-2.C 编译成汇编语言观察:

```
C>Tcc -S int24-2
Turbo C Version 2.0 Copyright (c) 1987, 1988 Borland International
```

```
int24-1.C:
Warning int24-2.C 5 : Parameter 'AX' is never used in function NewInt24
Warning int24-2.C 5 : Parameter 'BX' is never used in function NewInt24
Warning int24-2.C 5 : Parameter 'CX' is never used in function NewInt24
Warning int24-2.C 5 : Parameter 'DX' is never used in function NewInt24
Warning int24-2.C 5 : Parameter 'EX' is never used in function NewInt24
Warning int24-2.C 5 : Parameter 'DS' is never used in function NewInt24
Warning int24-2.C 5 : Parameter 'SI' is never used in function NewInt24
Warning int24-2.C 5 : Parameter 'DI' is never used in function NewInt24
Warning int24-2.C 5 : Parameter 'BP' is never used in function NewInt24
```

接着，瞧瞧 INT24-2.ASM 的清单：

```
            ifndef   ? ? version
            ?debug   macro
            endm
            endif
            ?debug   S "int24-2.c"
_TEXT       segment  byte public 'CODE'
DGROUP      group    _DATA,_BSS
            assume   cs:_TEXT,ds:DGROUP,ss:DGROUP
_TEXT       ends
_DATA       segment  word public 'DATA'
d@          label    byte
d@w         label    word
_DATA       ends
_BSS        segment  word public 'BSS'
b@          label    byte
b@w         label    word
            ?debug   C E944065A1809696E7432342D322E63
_BSS        ends
_TEXT       segment  byte public 'CODE'
;           ?debug   L 1
_NewInt24            proc    far
            push     ax
            push     bx
            push     cx
            push     dx
            push     es
            push     ds
            push     si
            push     di
            push     bp
            mov      bp,DGROUP
            mov      ds,bp
```

```
        mov     bp,sp
;       ?debug  L 4
        mov     word ptr [bp+16],0
@1:
;       ?debug  L 5
        pop     bp
        pop     di
        pop     si
        pop     ds
        pop     es
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        iret
_NewInt24       endp
_TEXT   ends
        ?debug  C E9
_DATA   segment word public 'DATA'
s@      label   byte
_DATA   ends
_TEXT   segment byte public 'CODE'
_TEXT   ends
        public  _NewInt24
        end
```

果然，Turbo C 就照着我们的意思直接修改堆栈里的 AX 寄存器了(movword prt[bp+16],0)。

照以上这个方法，我们也可将处理 Ctrl−Break 的中断 INT 23h 改写成如下形式，以避免程序在进行 I／O 时被用户中断：

```
void interrupt NewInt23()
{
}
```

7.5 驻留程序的实例

7.5.1 驻留计时器的编写

在本节中，我们开始利用 C 语言编写驻留程序。首先我们在文字模式下编写一个计时器，这个计时器会在屏幕右上角显示从运行本程序后到目前所经过的秒数。你可以按两边的 Ctrl+Alt 键切换声音的开与关，两个 Shift 键同时按下将会把秒数归零。让我们来一步一步地分析这个计时器程序 TIMER.C：

```
#include <stdio.h>
```

```
#include <dos.h>
#include <bios.h>

#define ON          1
#define OFF         0
#define RShift      0x0001
#define LShift      0x0002
#define Ctrl        0x0004
#define Alt         0x0008
#define SpecialKey (*((unsigned int far *)0x00400017))
#define VideoMode  (*((unsigned char far *)0x00400049))
```

程序开始时先包含所需要的头文件，并定义本程序将会用到的有关常量。其中Specialkey常数目的是取得BIOS数据区中的特殊按键的0040:0017及0040:0018两个字节。在内存中，字节数据是由低地址向高地址区存放，所以看起来会是像下图这个样子：

0040:0016	0040:0017	0040:0018	0040:0019

但是如果我们以整数的方式读出的话，会变成0040:0018被放在高字节而0040:0017被放在低字节，就像下图这个样：

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

0040:0018 高字节　　　　0040:0017 低字节

参考7.2.2节对这两个字节的介绍，若是要检查两边的Shift按键，就检查D0及D1；若是要检查INSERT按键，那么就必须检查D15(原0040:0018的D7)了。常量RShift，LShift，Ctrl及Alt就是利用这个原理来定义，作为检查该键之用的。

VideoMode常量则是用相同的方法取得BIOS数据区中记录的目前图像模式编号.

```
void interrupt new1c();
void interrupt (*old1c)();
void interrupt new09();
void interrupt (*old09)();
```

定义新旧中断的函数。程序中使用INT 09及INT 1C来检查按键，并利用INT 1C来激活驻留程序。

```
extern void far *_heapbase;

static unsigned count=0;
static int times=0;
static int tick_per_sec=18;
static int timer=0;
```

```
static unsigned ZeroKey  = RShift + LShift;
static unsigned SoundKey = Ctrl   + Alt   ;
static unsigned TsrSS, TsrSP;
static unsigned OldSS, OldSP;
static unsigned TsrBusy=0;
static unsigned sound_sw=ON;
```

这个程序中断用来定义程序中所要使用到的变量，其中：

* _heapbase	必须定义成外部变量，程序编译后Turbo C会自动为它赋上适当的值。
count	用来记录每一次滴答(tick)的计数。
tick_per_sec	记录每秒的滴答数，由于确实的滴答数应是每秒18.2次，而我们都用整数来计算，所以必须以 4 次 18 tick／sec 及 1 次 19 tick／sec 来矫正这项累积误差．times 就是用来计算这些次数之用．也就是当 times 不等于 5 的时候，count 每数到 18 秒就加 1 秒．这样才能编写出较准的计时器．
timer	记录目前的秒数。
Zerokey	将计时器归零的特殊按键。
Soundkey	切换声音开关的特殊按键。
TsrSS及TsrSP	用来保留驻留程序自身的堆栈指针。
OldSS及OldSP	用来保留前台程序的堆栈指针．
TsrBusy	用来防止驻留程序自身的重入。
sound_sw	声音的开关．

```
main()
{
    delay(0);

    disable();
    old1c  =  getvect(0x1c);
    setvect(0x1c, new1c);
    old09  =  getvect(0x09);
    setvect(0x09, new09);
    enable();

    TsrSS  =  _SS;
    TsrSP  =  _SP;
    puts("[[[*  Timer  *]]] install...");
    keep(0, FP_SEG(_heapbase)-_psp);
    puts("Can't install TSR program!");
}
```

主程序首先用新的 INT 09 及 INT 1C 取代旧的中断，在取代的过程中为了怕其它中断的干扰，所以使用 disable()函数禁止其它中断的产生.

接下来是保存堆栈指针，打印驻留程序设置的信息，接着就使用 keep()函数运行驻留. 如果驻留成功，那么就不会再返回了；若是驻留失败，将会再返回主程序中运行下一行 puts()，输出不能驻留的出错信息。

```
void interrupt new1c()
{
    ( * old1c)();

    disable();
    if(TsrBusy) return;
    TsrBusy++;
    enable();
```

运行新的 INT 1C，首先必须先运行旧的中断 INT 1C 以维持软件的兼容性，然后再检查 TsrBusy 变量看看是否已经正在运行本驻留程序. 如果是的话就立刻返回，以免造成自身的重复进入；如果没有，则把 TsrBusy 加 1，继续运行下去

```
OldSS = _SS,
OldSP = _SP;
_SS = TsrSS;
_SP = TsrSP;
```

进入驻留程序第一件事便是切换堆栈，我们可以借助虚拟变量_ss 及_sp 完成.

```
if((SpecialKey & SoundKey)= =SoundKey)
    sound_sw = (sound_sw) ? OFF : ON;
if((SpecialKey & ZeroKey)= =ZeroKey)
    timer = 0;
```

检查按键，看看是否为程序中指定的特殊按键；若是，则运行相对应的动作。

```
DoTsr();

_SS = OldSS;
_SP = OldSP;
TsrBusy--;
}
```

运行驻留程序，运行完毕后再切回原来的堆栈，然后将 TsrBusy 归零，允许第一次的激活。

```
void interrupt new09()
{
  ( * old09)();

  disable();
```

```
    if(TsrBusy) return;
    TsrBusy++;
    enable();

    OldSS = _SS;
    OldSP = _SP;
    _SS = TsrSS;
    _SP = TsrSP;

    if((SpecialKey & SoundKey)==SoundKey)
      sound_sw = (sound_sw) ? OFF : ON;
    if((SpecialKey & ZeroKey)==ZeroKey)
      timer = 0;

    _SS = OldSS;
    _SP = OldSP;
    TsrBusy--;
}
```

New09 函数和 New1C 函数类似，就不再多做说明了。

```
  static DoTsr()
{
  char str[10];
  int  i;
  unsigned segaddr;

  if(++count>=tick_per_sec) {
    count=0;
    if(sound_sw) {
        sound(3000);
        delay(5);
        nosound();
  }
  timer = (++timer>999) ? 0 : timer;
  if(++times==5) {
      times=0;
      tick_per_sec=19;
  }
  else tick_per_sec=18;
}
strcpy(str, "Timer:");
str[6] = '0' + timer / 100;
str[7] = '0' + (timer%100) / 10;
str[8] = '0' + timer%10;
```

```
if(VideoMode= =3) segaddr = 0xb800;
else              segaddr = 0xb000;
for(i=0; i<9; i++) {
    pokeb(segaddr, (71+i) * 2, str[i]);
    pokeb(segaddr, (71+i) * 2+1, 0x70);
  }
}
```

以上这段驻留程序的主程序都是使用直接填写内存的方式、所使用的函数也不曾用到DOS的功能调用，可以放心运行，并不会造成DOS的重入。

本函数基本上分成计数和显示两大部分。计时部分已在前面提过，不再重述，显示部分主要分成几个步骤：

1. 准备一个字符数组str[0]，并将"Timer"字符串填入。
2. 将数值timer化成字符串分别依百位、十位、个位的顺序填入str[6]，str[7]及str[8]。
3. 检查目前的屏幕模式，若为彩色文字模式(VideoMode= =3)，则显示内存位置应从B800H开始，否则从B000H开始。
4. 用一个循环将字符及属性值填入即可。

7.5.2 驻留小时钟

接下来要介绍的这个程序稍微复杂一点。CLOCK.C的主要功能是在屏幕的右上角显示目前系统的时间，不只是在文字模式下可以看得见，即使屏幕进入绘图模式后（包括图像模式11h、12h及13h）bn看得见这个时钟；如果在256色的绘图模式下，甚至可以看到忽明忽暗的数字显示。同时，为避免驻留程序载入后除非重新开机无法去除的缺点，这个程序特别增加了一个热键用来去除驻留程序。

这个程序使用到了3个中断，分别是INT 09、INT 1C，INT 28。其中INT 1C是驻留的主程序，负责读取并显示时间，并且检查用户是否按下去了去除驻留程序的热键(左右Shift同时按下)；若有，则运行去掉驻留程序的动作。INT 09及INT 28则只是单纯地负责检查热键并去掉驻留程序的任务。

在去除驻留程序的部分，INT 09，INT 1C及INT 28有着许多的不同，因为去除驻留程序的任务必须使用到DOS的功能调用，如果在INT 09及INT 1C中运行去除，则必须确定目前并没有DOS的功能调用，以免造成DOS的重入。如果是在INT 28内运行去除驻留程序的任务，就没有这层顾虑了；因为前文曾经提过，在INT 28中可以放心地使用DOS 01～0C以外的功能调用，而不会导致DOS的重入。

接着，我们就逐段来介绍CLOCK.C：

```
#include <stdio.h>
#include <dos.h>
#include <bios.h>

#define ON       1
#define OFF      0
#define RShift   0x0001
```

```c
#define LShift       0x0002
#define SpecialKey (*((unsigned int far *)0x00400017))
#define VideoMode  (*((unsigned char far *)0x00400049))

void interrupt new09();
void interrupt (*old09)();
void interrupt new1c();
void interrupt (*old1c)();
void interrupt new28();
void interrupt (*old28)();

extern void far *_heapbase;
extern unsigned _heaplen=0x800;
extern unsigned _stklen=0x800;

static unsigned RemoveKey = RShift + LShift;
static unsigned TsrSS, TsrSP, TsrPSP;
static unsigned OldSS, OldSP;
static unsigned TsrBusy=0;
static char far *DosBusy;
union REGS r;
struct SREGS sr;
```

和前一个程序 TIMER.C 不同的是：本程序多增加了两个主要的驻留程序编写所需的变量 TsrPSP 及 * DosBusy 变量。TsrPSP 主要的功能是存储驻留程序的 PSP。在这个程序例中，我们将使用 TsrPSP 来释放驻留程序所占用的内存。 * DosBusy 变量则记录目前是否处于 DOS 的功能调用状态下，如果这个值等于 0 的话表示目前 DOS 是处于闲置状态，驻留程序可以使用任何 DOS 的功能调用。

```c
main()
{
  delay(0);

  disable();
  old09 = getvect(0x09);
  setvect(0x09, new09);
  old1c = getvect(0x1c);
  setvect(0x1c, new1c);
  old28 = getvect(0x28);
  setvect(0x28, new28);
  enable();

  TsrSS = _SS;
  TsrSP = _SP;
  TsrPSP = _psp;
  r.h.ah = 0x34;
```

```
  intdosx(&r, &r, &sr);
  DosBusy = MK__FP(sr.cs, r.x.bx);

  puts("[[[ *  Clock  * ]]] install...");
  keep(0, FP__SEG(__heapbase)-__psp);
  puts("Can't install TSR program!");
}
```

在主程序中（尚未进入驻留之前）先行取得所有需要的参量，包括 TsrSS、TsrSP、TsrPSP 及 DosBusy，其中 DosBusy 的取得是使用 DOS 未公开的 34h 调用，读者应该不会觉得陌生才对。

```
void interrupt new1c()
{
  (*old1c)();

  disable();
  if(TsrBusy) return;
  TsrBusy++;
  enable();

  OldSS = __SS;
  OldSP = __SP;
  __SS = TsrSS;
  __SP = TsrSP;

  if(!(*DosBusy)) {
```

这段程序中最主要的部分是检查是否在 DOS 的忙碌状态下；如果不是，就可以继续下一步热键检查，看看是否要运行去除驻留程序的任务；如果 DOS 处于忙碌状态下（*DodBusy 不等于 0），那么按键根本就不用检查了，直接运行本中断其它的任务。

```
if((SpecialKey & RemoveKey)==RemoveKey)
{
      disable();
      setvect(0x09, old09);
      setvect(0x1c, old1c);
      setvect(0x28, old28);
      __ES = TsrPSP;
      __AH = 0x49;
      ___int___(0x21);
      __ES = peek(TsrPSP, 0x2c);
      __AH = 0x49;
      ___int___(0x21);
      enable();
    }
}
```

当 DOS 处于闲置的状态下时，就检查用户是否按下去除驻留程序的热键，如果有的话就运行去除驻留程序的动作。一般而言，去除驻留程序至少必须包括下面 2 个步骤：

1. 恢复原中断矢量。

2. 释放驻留程序所占用的内存。

恢复中断矢量没什么问题，而释放内存部分可就必须稍做说明了。

在 DOS 下运行一个程序，通常 DOS 都会分配 2 块内存给运行的程序，其中一块放置程序本身，而另一块放置这个程序的环境变量。那么 DOS 如何来管理这些分配的内存呢？DOS 在每一块分配出去的内存以及尚未使用的内存之前，都会有一个 16 个字节大小的内存控制区块 MCB(Memory Control Block)，其每个字节的作用如下表所示：

表 7-5　MCB 各字节说明

起始位移	长度(Byte)	用　　途
0	1	识别码，若是最后一个 MCB 则为 5Ah，否则为 4Dh。
1	2	拥有此块内存的程序的 PSP 段地址，若为 0000 表示这是一块尚未使用的内存。
3	2	此块内存的长度，以段(Paragraph)为单位。
5	11	保留

所以若以 MCB 的角度来看，我们若运行一个程序，这个程序在内存中应是像下图这个样子排列。

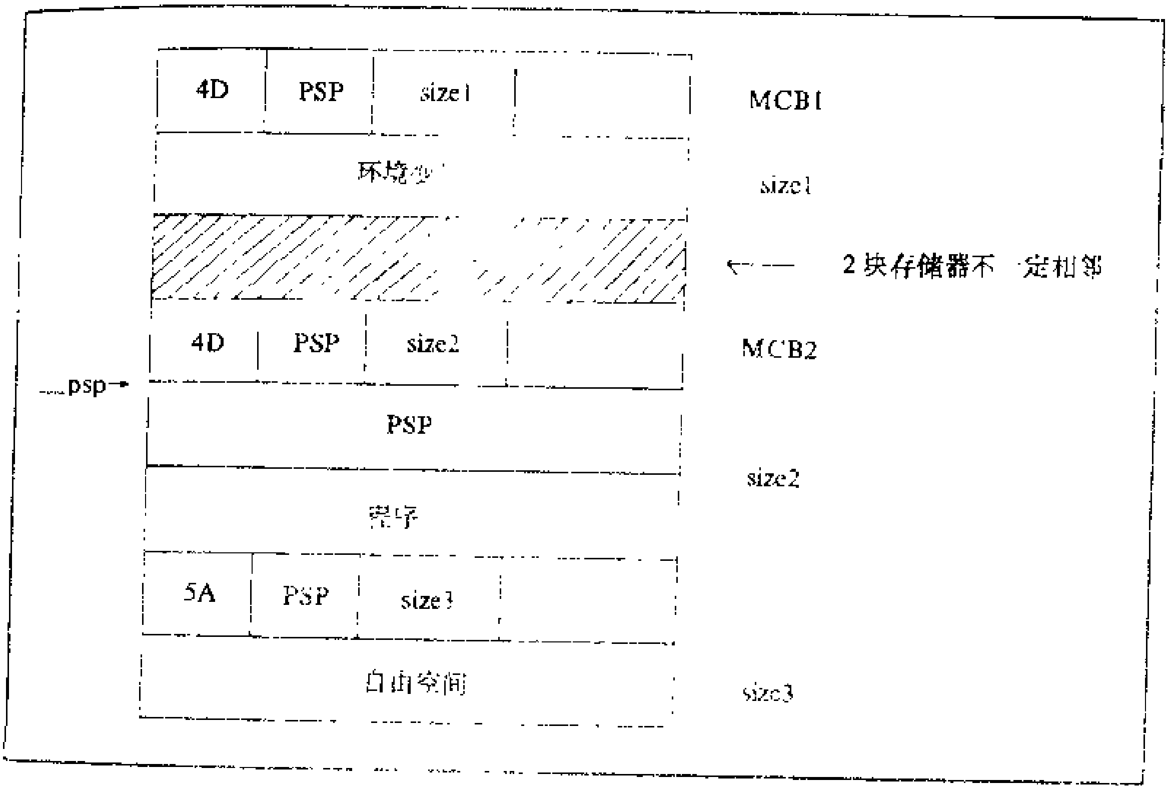


图 7-4　程序的 MCB

一般的程序在运行完毕后，DOS 会自动把这两块分配出去的内存收回供后面要运行的程序使用；驻留程序则不然，要由我们自己动手将两块内存释放，释放内存的功能调用 49h 列表如下：

表 7-6 DOS 内存释放

功 能	：	释放DOS所分配的内存区块
输入值	：	AH = 49H
		ES = 要释放区块的段地址
返回值	：	如果进位标志(Carry Flag)清除表示释放成功，否则AX返回错误码。
		AX = 07 ：MCB已遭破坏
		AX = 09 ：不正确的内存区块地址

我们已经知道 PSP 地址就是存放程序的内存区块的起始地址，至于环境变量内存区块的起始地址则可以在 PSP 中偏移位 2Ch 处找到，只要这两块内存都释放完毕，驻留程序也就完全去除了。

另外这个程序段中我们使用了不同于以往的方式来调用 DOS。你也可以用 geninterrupt()函数或 int86x()函数，只不过使用__int__的方式比较直接，可以产生较简洁的程序码。但是使用虚拟变量__AX，__ES 等来直接设置寄存器要特别小心，因为 Turbo (会和) AX 或其他寄存器作为暂时变量，这时 AX 原来的值可能会被破坏了。所以如果你实在没有把握，不妨在命令行编译下使用-s 参量将源程序转译成汇编语言检查一番。

```
    DoTsr();

    __SS = OldSS;
    __SP = OldSP;
    TsrBusy--;
}
```

这段语句的任务是运行主程序，然后切回原来的堆栈。

```
void interrupt new09()
{
  (*old09)();

  disable();
  if(TsrBusy) return;
  TsrBusy++;
  enable();

  OldSS = __SS;
  OldSP = __SP;
  __SS = TsrSS;
  __SP = TsrSP;
```

```
    if(!( * DosBusy)) {
      if((SpecialKey & RemoveKey) = = RemoveKey)
      {
        disable();
        setvect(0x09, old09);
        setvect(0x1c, old1c);
        setvect(0x28, old28);
        __ES = TsrPSP;
        __AH = 0x49;
        ___int___(0x21);
        __ES = peek(TsrPSP, 0x2c);
        __AH = 0x49;
        ___int___(0x21);
        enable();
      }


  __SS = OldSS;
  __SP = OldSP;
  TsrBusy--;
}
```

new09 除了未运行 DoTsr 外，其余都和 new1c 一样.

```
void interrupt new28()
{
    ( * old28)();

    disable();
    if(TsrBusy) return;
    TsrBusy++;
    enable();

    OldSS = __SS;
    OldSP = __SP;
    __SS = TsrSS;
    __SP = TsrSP;

  if((SpecialKey & RemoveKey) = = RemoveKey)
  {
    disable();
    setvect(0x09, old09);
    setvect(0x1c, old1c);
    setvect(0x28, old28);
    __ES = TsrPSP;
    __AH = 0x49;
```

```
        __int__(0x21);
        _ES = peek(TsrPSP, 0x2c);
        _AH = 0x49;
        __int__(0x21);
        enable();
    }

    _SS = OldSS;
    _SP = OldSP;
    TsrBusy--;
}
```

new28 也和 new09 一样，只是少了一项 DOS 的忙碌标志检查。

```
static DoTsr()
{
    char str[10];
    int  i, j, k;
    union REGS r;
    unsigned hour,min,sec,
```

这段程序码定义了 DoTsr 函数所需的变量。请注意：由于在进入本函数之前就已经切换过堆栈了，所以在此定义的局部变量便推入驻留程序本身的堆栈。而且由于是切换过堆栈后才进入本函数，所以本函数的返回地址也都在驻留程序的堆栈之中。

Turbo C 会将自动变量（即局部变量）堆栈中以备使用，假设我们在运行切换堆栈的函数（如 new1c）中定义自动变量，那么在程序码运行之前这些自动变量便都无条件被推往旧的堆栈（前台程序的堆栈）中，这样待我们切换堆栈后，程序就找不到原来定义的变量了。这就是为什么我们会将一些 new09，new1c 及 new28 函数中使用到的变量定义成全局变量的原因。因为全局变量是放在另一块地方，而不是在堆栈中使用。

```
r.h.ah = 2;
int86(0x1a, &r, &r);
hour  = ((r.h.ch & 0xf0)>>4) * 10 + (r.h.ch & 0xf);
min   = ((r.h.cl & 0xf0)>>4) * 10 + (r.h.cl & 0xf);
sec   = ((r.h.dh & 0xf0)>>4) * 10 + (r.h.dh & 0xf);

str[0] = hour / 10 + '0';
str[1] = hour%10 + '0';
str[2] = ':';
str[3] = min / 10 + '0';
str[4] = min%10 + '0';
str[5] = ':';
str[6] = sec / 10 + '0';
str[7] = sec%10 + '0';
```

这段是读取时间，并将它转换成这字符串的过程。为了避免 DOS 的重入问题，在此

我们使用 BIOS 的 1Ah 中断，其功能说明如下：

表 7-7 BIOS 读取实时时钟功能调用列表

功 能	:	读取即时时钟
输入值	:	AH = 02
		INT 1AH
返回值	:	如果进位标志(Carry Flag)清除，表示读取成功，否则失败。
		CH = 小时，以BCD的格式返回。
		CL = 分钟，以BCD的格式返回。
		DH = 秒数，以BCD的格式返回。

由于寄存器所返回的值都是以 BCD 的格式表示，所以我们还必须做适当的转换才能使用。所谓 BCD(Binary Coded Decima)格式，就是以 4 个位来表示 1 个十进位 0～9，至于 10～15(1016～1116)则忽略不用，所以若有 1 个十进位的数字 58，则 BCD 表示成：

```
0 1 0 1     1 0 0 0
└─────┘     └─────┘
   5           8
```

假如我们要把 BCD 格式转换成十进位，只要将左边的数字 5 乘以 10 再加右边的 8 就得到 58 了。

上面这段程序为了清楚地说明起见，特别将转换成十进位的数字的过程列出，聪明的读者一定已经发现，其实我们可以直接将 BCD 转换成字符串，写成如下面这个样子：

```
r.h.ah  = 2;
int86(0x1a,&,&r);
str[0]  = ((r.h.ch & 0xf0)> >4) + '0';
str[1]  =  (r.h.ch & 0x0f)      + '0';
str[2]  = ':';
str[3]  = ((r.h.cl & 0xf0)> >4) + '0';
str[4]  =  (r.h.cl & 0x0f)      + '0';
str[5]  = ':';
str[6]  = ((r.h.dh & 0xf0)> >4) + '0';
str[7]  =  (r.h.dh & 0x0f)      + '0';
```

是不是简单多了！接下来就是显示的部分了：

```
if((VideoMode= =3) | | (VideoMode= =7)) {
  for(i=0; i<8; i++) {
      pokeb((VideoMode= =3) ? 0xb800 : 0xb000, (72+i) * 2, str[i]);
      pokeb((VideoMode= =3) ? 0xb800 : 0xb000, (72+i) * 2+1, 0x70);
  }
}
```

这段程序首先检查是不是文字模式，如果是的话就用上一节介绍的方法将字符的各字符及属性写在屏幕的右上角。

```
else if((VideoMode==0x10) || (VideoMode==0x12))
  for(j=0; j<8; j++)
     for(j=0; j<8; j++)
        pokeb(0xa000, j*80+72+i, peekb(0xf000, 0xfaoe+str[i]*8+j));
```

再来看看是否为彩色 640×350 或 640×480 的绘图模式；若是，就到 BIOS 的 F000:FA6E 处去取字形数据将它写到屏幕右上角。在此我们缺省是第 0 页显示，如果读者有兴趣，亦可加入显示页的判断，直接修改段地址（0xa000 或 0xa800）即可。由于我们没有做颜色的设置，所以你可能会看到白色的字，若是要设置颜色，你可以加上：

```
outportb(0x3c4,2);
outportb(0x3c5,color);
```

其中 color 就是你要的字形颜色，不过这可能会影响到前台程序色彩的操作。

```
else if(VideoMode==0x13)
for(i=0; i<8; i++)
      for(j=0; j<8; j++)
          for(k=0; k<8; k++)
              if((peekb(0xf000, 0xfa6e+str[i]*8 + j)>>(7-k)) & 1)
                  pokeb(0xa000, j*320+256+i*8+k, 26-j);
              else
                  pokeb(0xa000, j*320+256+i*8+k, 0);
}
```

如果是 256 色模式，显示方法就有所改变了。最大的变化在于 256 色模式是 1 个字节表示一个像素点，所以在从 F000:FA6E 处取得一个字节的数据，并不能像 16 色模式一样直接填入显示内存，而必须一个位一个位拆开，再依其值（1 或 0）填入相对应的该点位置。其中在写入颜色部分，我们故意选取第 26 号开始的连续 8 个颜色，这样就可以看到层次分明的显示效果了。

现在就运行看看 CLOCK.EXE，并检查它所占用的内存：

```
C>clock
[[[*  Clock  *]]] install...

C>mem/c
```

Conventional Memory:

Name	Size in Decimal	Size in Hex
MSDOS	13568(13,3K)	3500
HIMEM	1184(1.2K)	4A0
EMM386	8400(8.2K)	20D0
COMMAND	2624(2.6K)	A40
CLOCK	71280(69.6K)	11670 ☆
FREE	64(0.1K)	40
FREE	558000(544.9K)	883B0
Total FREE:	55864(545.0K)	

从以上的内存使用状态中可以发现：小小一个时钟程序竟然占掉了 69.6K，到底怎么一回事呢？原来 Turbo C 缺省在 Small 及 Medium 存储模式下分配 64K 的内存给数据段使用，那么数据段包括哪些数据呢？数据段的大小计算方式如下：

数据段大小 = 全局数据 + 近程堆栈 + 堆栈

在缺省的情况下（亦即没有指定堆大小的情况），整个数据段之大小为 64K，假使全部保留成为驻留程序，程序占用的内存自然大得不像话了。所以我们只需在程序前端加入下列这行命令，重新编译运行：

```
extern unsigned _heaplen=0x800;
```

重新编译后，再检查内存大小。

Conventional Memory:

Name	Size in Decimal	Size in Hex
MSDOS	13568(13,3K)	3500
HIMEM	1184(1.2K)	4A0
EMM386	8400(8.2K)	20D0
COMMAND	2624(2.6K)	A40
CLOCK	12960(12.7K)	32A0 ☆
FREE	64(0.1K)	40
FREE	616320(601.9K)	96780
Total FREE:	6163384(601.9K)	

占用的内存只剩下 12.7K。由于我们使用的堆栈也不大，因此也可以将原来的 4K（缺省值）减为 2K。

```
extern unsigned _stklen=0x800;
```

再检查内存大小：

Conventional Memory:

Name	Size in Decimal	Size in Hex
MSDOS	13568(13,3K)	3500
HIMEM	1184(1.2K)	4A0
EMM386	8400(8.2K)	20D0
COMMAND	28624(2.6K)	A40
CLOCK	10912(12.7K)	2AA0 ☆
FREE	64(0.1K)	40
FREE	618369(603.9K)	96F80
Total FREE:	618432(601.9K)	

正如我们所料，占用内存又少了 2K。

到此为止，相信你对_stklen 及_heaplen 两变量的用法应该有所了解了吧！这两个变量对一般的程序也许并没有什么重要性，然而却对于内存要求严苛的驻留程序极为重要。两个值的大小并没有固定，全看程序的大小及使用到变量或内存分配的多寡而定，你可以自行试着调整看看。

7.5.3 驻留屏幕区块放大程序

还记不记得前几章所介绍的荧幕放大程序？这一节我们就把 ZOOM.C 转换常驻程序 ZOOMTSR.C，让我们可以在各种不同的场合下调出放大程序，使这个程序有更宽广的应用。

当然，照例除了使驻留程序能够驻留之外，我们也为这个程序加上几项，编写坚固耐用的驻留程序所需的技术。这些技术包括防止重复载入驻留程序以及利用命令行来去除驻留程序。

```
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include "supervga.h"
#include "key.h"
int Screen__Width;
int Screen__Height;
int Char__Height;
int Screen__Rows;
int Window__Width =2;
int Window__Height =2;
```

这些变量都是运行屏幕区块放大函数所需的，你应该不会陌生吧!

```
void interrupt new09();
void interrupt (*old09)();
void interrupt new1c();
void interrupt (*old1c)();
void interrupt new28();
void interrupt (*old28)();
void interrupt (*old60)();
void interrupt new61();
void interrupt (*old61)();
int TsrInMem(void);
```

和上一个程序 CLOCK.C 比较，这个程序额外地使用了 INT 60h 及 INT 61h 两个中断。查阅 PC BIOS／DOS 的相关参考书籍可以知道：PC 保留了 60h～67h 等 8 个中断矢量供用户自行运用，平常状态下，这 8 个中断矢量的值都被填入 0。我们便利用 INT 60h 来做为驻留程序的识别码，而 INT 61h 去除驻留程序的服务子程序，至于用法将于后面详述。另外，TsrInMem 函数的功能为检查驻留程序是否已经驻留在内存中了，假如是的话返回 1，否则返回 0。

```
void Action(void);
void Zoom(int, int, int, int);
void Readinfo(void);
```

这三个函数都是屏幕放大程序的主要函数，在此不多加说明；如果不清楚，请参考本书第五章。

```
extern void far * _heapbase;
static unsigned HotKey =RShift + LShift;
static unsigned TsrSS, TsrSP, TsrPSP;
static unsigned OldSS,OldSP;
static unsigned TsrBrsy=0;
static char far * DosBusy;
union REGS r;
struct SREGS sr;
unsigned char id[4] ={0xf1,0xf2,0xf3,0xf4};
```

这些变量中比较特殊的是 id[4]，这是本驻留程序的识别码。程序每次被载入运行时，一定先到 INT 60h 的中断矢量处检查是否存在有这个识别码，由此判断驻留程序是否已经载入，以避免内存中同时有两份以上相同的驻留程序。

此外如果要节省内存的使用，你也可以把 _stklen 及 _heaplen 变量放在 * _heapbase 变量后，你只要指定适当的大小即可。

```
main(argc,argv)
int argc;
char * argc[];
{
        deltay(0);
        if(argc>1)
            if(!strnicmp(argv[1],"remove",5)){
                if(!TsrInMem()) {
                    put("There is no Zoom Utility in memory!");
                    exit(0);
                }
                else {
                    __int__(0x61);
                    puts("Zoom Utility already removed!");
                    exit(0);
                }
}
```

这段主程序一开始便检查用户是否输入命令行参量；若有，则检查第一个命令行数是否为 Remove 开头（其它的忽略），若是，表示用户打算撤除常驻程序，接着便进入撤除驻留程序的步骤。但是在进入撤除步骤之前得先检查驻留程序是否已经在内存中了，若有才运行撤除，否则贸然撤除一个不存在的程序，恐怕会造成错误。

这次我们把撤除驻留程序任务交由中断 INT 61 来完成。为什么我们会选择以中断服务子程序的方式完成而不直接将撤除的有关程序码写在这里，或是以函数调用的方式完成呢？原因很简单，因为后来运行的程序和已经驻留的程序虽然在磁盘上是同一个执行文

件，但是在内存中却是两份不同的拷贝．所以后来运行的程序则无法得知已驻留程序的PSP，因此无法撤除已驻留的程序，只能用驻留程序自己准备好的撤除中断 61h，通知已驻留的程序把自己撤掉．请看下面的图解：

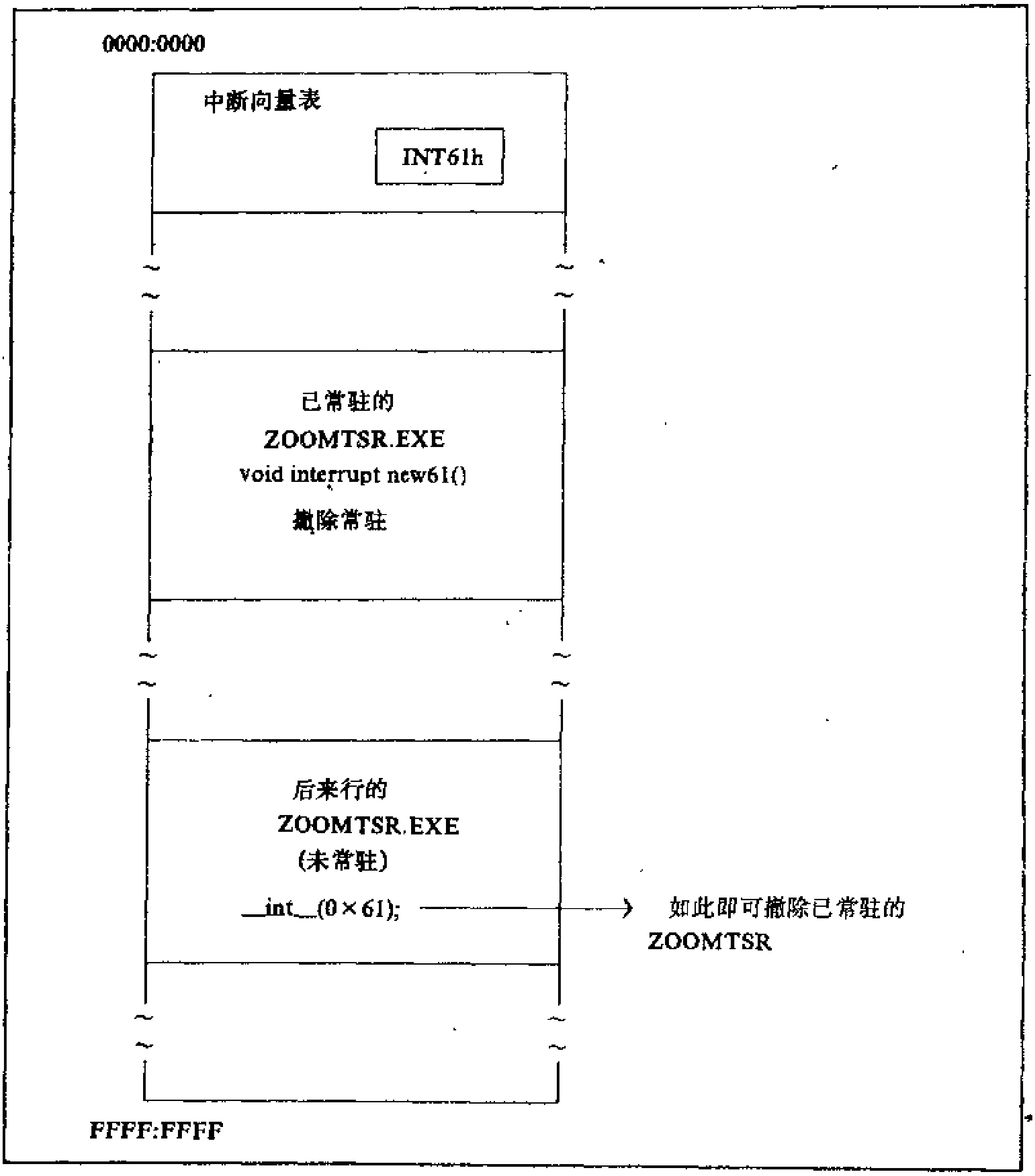


图 7-5　ZOOMTSR 在内存中的位置

请特别注意：在调用 INT 61h 撤除原有驻留程序后，要立刻使用 exit 函数结束程序，否则继续运行下去又把自己驻留了，那岂不是白努力一场了吗？

```
    if(TsrInMen()){
        puts("Zoom Utility already resident!");
        exit (0);
    }
```

如果用户并未指定 REMOVE 参量表示要进入驻留程序，此时，同样地先使用 TsrInMen 函数检查是否已驻留过。如果有，则显示出系统已驻留的信息，并结束程序。

```
  old60=getvect(0x60);
  movedata(FP_SEG(id),FP_OFF(id),0,0x180,4);
```

程序驻留之前，先在 INT 60h 的中断矢量处填入我们驻留程序的识别记号 id。当然，在填识别记号之前，还是要保存原 INT 60h 的中断矢量，以维持兼容性。

```
    disable();
    old09=getvect(0x09);
    setvect(0x09,new09);
    old1c=getvect(0x1c);
    setvect(0x1c,new1c);
    old28=getvect(0x28);
    setvect(0x28,new28);
    old61=getvect(0x61);
    setvect(0x61,new61);
    enable();
    TsrSS = _SS;
    TsrSP = _SP;
    TsrPSP = PSP;
    r.h.ah =0x34;
    intdosx(&r,&r,&sr);
    DosBusy=MK_FP(sr.es,r.x.bx);
    put("[[[ * Zoom Utility * ]]] install...");
    keep(0,FP_SEG(_headbase)-_psp);
    puts("Can't install TSR program!");
  }
```

以上这些步骤相信你都已经很熟悉了，不再多加说明，现在就把 main 函数流程做一整理，供各位读者参考：

```
  void interrupt new1c()
  {
      ( * old1c)();

      disable();
      if(TsrBusy) return;
      TsrBusy ++;
      enable();
```

```
    OldSS = _SS;
    OldSP = _SP;
    _SS = TsrSS;
    _SP = TsrSP;

    if((SpecialKey &HotKey)= =HotKey) Action();

    _SS=OldSS;
    _SP=OldSP;
    TsrBusy--;
}
```

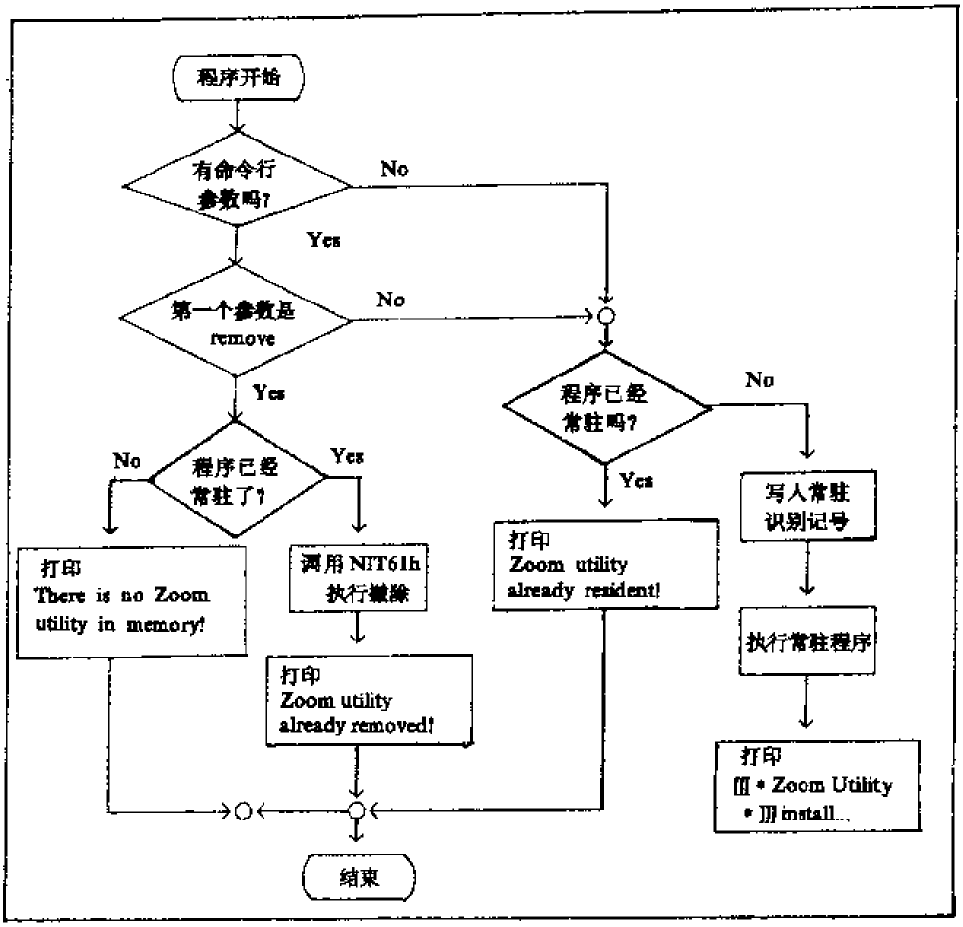


图 7-6　ZOOMTSR.C 程序的框图

由于 Action 中并未使用任何 DOS 功能调用，所以可以放心地在 new1c 中激活 Action，不必考虑 DosBusy 变量。

```
void interrupt new09()
{
    (*old09)();

    disable();
    if(TsrBusy) return;
```

```
    TsrBusy++;
    enalbe();

    OldSS=SS;
    OldSP=SP;
    __SS=TsrSS;
    __SP=TsrSP;

    if((SpecialKey &HotKey)= =HotKey) Action():

    __SS=OldSS;
    __SP=OldSP;
    TsrBusy—;
    }

void interupt new28()
{
    (*old28)();

    disable();
    if(TsrBusy) return;
    TsrBusy++;
    enable();

    OldSS =__SS;
    OldSP =__SP;
    __SS= TsrSS;
    __SP= TsrSP;

    if(SpecialKey & HotKey)= =HotKey) Action();

    __SS= OldSS;
    __SP= OldSP;
    TsrBusy—;
}
```

new09，new08 和 new1c 一样，无需多做介绍。

```
void interrupt new61()
{
    disable();
    setvect(0x09,old09);
    setvect(0x1c,old1c);
    setvect(0x28,old28);
    setvect(0x61,old60);
```

```
        _ES  = TsrPSP;
        _AH  = 0x49;
        _int_(0x21);
        _ES  = peek(TsrPSP,0x2c);
        _AH  = 0x49;
        _int_ (0x21);
        enable();
}
```

new61 中断函数运行撤除驻留程序的操作，这样的写法应不陌生吧！

```
int TsrInMem()
{
        unsigned char Int60[4];

        movedata(0,0x180,FP_SEG(Int60),FP_OFF(Int60),4);
        if(!strncmp(Int60,id,4)) return 1;
        else return 0;
}
```

这个函数的功能是检查 INT 60h 的中断矢量的值是否和驻留程序的识别码相同，若是则返回 1，否则返回 0。

接下来这段程序码便是和第五章所介绍的相同，在此就不多做说明。现在将列出完整程序清单供读者参考。

```
/* ZOOMTSR.C */

#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include "supervga.h"
#include "key.h"

#define RShift  0x0001
#define LShift  0x0002
#define SpecialKey (*((unsigned int far *) 0x00400017))

static unsigned Screen_Width;
static unsigned Screen_Height;
static unsigned Char_Height;
static unsigned Screen_Rows;
static unsigned Window_Width = 2;
static unsigned Window_Height= 2;

void interrupt new09();
void interrupt (*old09)();
void interrupt new1c();
```

```
void interrupt ( * old1c)();
void interrupt new28();
void interrupt ( * old28)();
void interrupt ( * old60)();
void interrupt new61();
void interrupt ( * old61)(),
int  TsrInMem(void);
void Action(void);
void Zoom(int, int, int, int);
void Readinfo(void);

extern void far * _heapbase;

static unsigned HotKey = RShift + LShift;
static unsigned TsrSS, TsrSP, TsrPSP;
static unsigned OldSS, OldSP;
static unsigned TsrBusy=0;
static char far * DosBusy;
union REGS r;
struct SREGS sr;
unsigned char id[4] = {0xf1, 0xf2, 0xf3, 0xf4};

main(argc, argv)
int argc;
char * argv[];
{
      delay(0);
      if(argc>1)
        if(!strnicmp(argv[1], "remove", 5)) {
            if(!TsrInMem()) {
                  puts("There is no Zoom Utility in memory!");
                  exit(0);
                }
                else {
                ____int____(0x61);
                puts("Zoom Utility already removed!");
                exit(0);
                }
        }

      if(TsrInMem()) {
         puts("Zoom Utility already resident!");
         exit(0);
      }
```

```
        old60 = getvect(0x60);
        movedata(FP_SEG(id), FP_OFF(id), 0, 0x180, 4);
        disable();
        old09 = getvect(0x09);
        setvect(0x09, new09);
        old1c = getvect(0x1c);
        setvect(0x1c, new1c);
        old28 = getvect(0x28);
        setvect(0x28, new28);
        old61 = getvect(0x61);
        setvect(0x61, new61);
        enable();

        TsrSS = _SS;
        TsrSP = _SP;
        TsrPSP = _psp;
        r.h.ah = 0x34;
        intdosx(&r, &r, &sr);
        DosBusy = MK_FP(sr.es, r.x.bx);

        puts("[[[ *  Zoom Utility  * ]]] install...");
        keep(0, FP_SEG(_heapbase)-_psp);
        puts("Can't install TSR program!");
}

void interrupt new1c()
{
        ( * old1c)();

        disable();
        if(TsrBusy) return;
        TsrBusy++;
        enable();

        OldSS = _SS;
        OldSP = _SP;
        _SS = TsrSS;
        _SP = TsrSP;

        if((SpecialKey & HotKey)==HotKey)  Action();

        _SS = OldSS;
        _SP = OldSP;
        TsrBusy--;
}
```

```
void interrupt new09()
{
        ( * old09)();

        disable();
        if(TsrBusy) return;
        TsrBusy++;
        enable();

        OldSS = _SS;
        OldSP = _SP;
        _SS = TsrSS;
        _SP = TsrSP;

        if((SpecialKey & HotKey)==HotKey)
              Action();

        _SS = OldSS;
        _SP = OldSP;
        TsrBusy--;
}

void interrupt new28()
{
        ( * old28)();

        disable();
        if(TsrBusy) return;
        TsrBusy++;
        enable();

        OldSS = _SS;
        OldSP = _SP;
        _SS = TsrSS;
        _SP = TsrSP;

        if((SpecialKey & HotKey)==HotKey)   Action();

        _SS = OldSS;
        _SP = OldSP;
        TsrBusy--;
}

void interrupt new61()
{
        disable();
        setvect(0x09, old09);
```

```
        setvect(0x1c, old1c);
        setvect(0x28, old28);
        setvect(0x60, old60);
        setvect(0x61, old61);
        _ES = TsrPSP;
        _AH = 0x49;
        __int__(0x21);
        _ES = peek(TsrPSP, 0x2c);
        _AH = 0x49;
        __int__(0x21);
        enable();
}

int TsrInMem()
{
        unsigned char Int60[4];

        movedata(0, 0x180, FP_SEG(Int60), FP_OFF(Int60), 4);
        if(!strncmp(Int60, id, 4)) return 1;
        else return 0;
}

void Action()
{
    int key;
    int quit=0;
    int xscale=2;
    int yscale=2;
    int x=0, y=0;
    int ss, sp;
     unsigned maxx, maxy;
     sound(800); delay(100);
     sound(1200); delay(100);
     nosound();

  Readinfo();
  while(!quit) {
   key = bioskey(0);
   maxx = Screen_Width - Window_Width * (xscale + 1) -1;
   maxy = Screen_Height - Window_Height * (yscale +1) -1;
  switch(key) {
   case PGUP : Window_Width = (++Window_Width>20) ? 20 : Window_Width;
               break;
   case PGDN : Window_Width = (--Window_Width< 2) ?  2 : Window_Width;
             break;
```

```
      case HOME : Window_Height= (++Window_Height>20) ? 20 : Window_Height;
                  break;
      case END  : Window_Height= (--Window_Height< 2) ?  2 : Window_Height;
                  break;
      case UP   : y = (--y<0) ? 0 : y;
                  break;
      case DOWN : y = (++y>maxy) ? maxy : y;
                  break;
      case LEFT : x = (--x<0) ? 0 : x;
                  break;
      case RIGHT: x = (++x>maxx) ? maxx : x;
                  break;
      case PLUS : yscale = (++yscale>7) ? 7 : yscale;
                  break;
      case MINUS: yscale = (--yscale<1) ? 1 : yscale;
                  break;
      case INS  : xscale = (++xscale>7) ? 7 : xscale;
                  break;
      case DEL  : xscale = (--xscale<1) ? 1 : xscale;
                  break;
      case ESC  : quit=1;
                  break;
     }
     Zoom(xscale, yscale, x, y);
   }
   Zoom(0, 0, x, y);
}

void Readinfo()
{
  union REGS r;

  r.h.ah = 0xf;
  int86(0x10, &r, &r);
  switch(r.h.al)
  {
     case 0x10 : Screen_Width  = 80;
                 Screen_Rows   = 25;
                 Char_Height   = 14;
                 break;
     case 0x12 : Screen_Width  = 80;
                 Screen_Rows   = 30;
                 Char_Height   = 16;
                 break;
```

```
        case 0x13 :  Screen_Width  = 80;
                     Screen_Rows   = 45;
                     Char_Height   = 8;
                     break;
        case 0x29 :  Screen_Width  = 100;
                     Screen_Rows   = 37;
                     Char_Height   = 16;
                     break;
        case 0x2d :  Screen_Width  = 80;
                     Screen_Rows   = 25;
                     Char_Height   = 14;
                     break;
        case 0x2e :  Screen_Width  = 80;
                     Screen_Rows   = 30;
                     Char_Height   = 16;
                     break;
        case 0x30 :  Screen_Width  = 100;
                     Screen_Rows   = 37;
                     Char_Height   = 16;
                     break;
        case 0x37 :  Screen_Width  = 128;
                     Screen_Rows   = 48;
                     Char_Height   = 16;
                     break;
  }
  Screen_Height= Char_Height * Screen_Rows;
}

void Zoom(int xscale, int yscale, int sx, int sy)
{
    unsigned int code=0;
    int ex, ey;
    int maxx, maxy;

    xscale = (xscale<0) ? 0 : xscale;      /* force xscale into range 0 to 7 */
    xscale = (xscale>7) ? 7 : xscale;
    yscale = (yscale<0) ? 0 : yscale;      /* force yscale into range 0 to 7 */
    yscale = (yscale>7) ? 7 : yscale;

    code |= xscale << 4;  /* set x zoom factor */
    code |= yscale;       /* set y zoom factor */
    code |= 0x80;         /* zoom enable       */
    code = code << 8;
    code |= 6;
    outport(0x3c4, code); /* set zoom factor and enable zoom */
```

```
maxx = Screen_Width - Window_Width * (xscale + 1) -1;
maxy = Screen_Height - Window_Height * (yscale +1) -1;

sx = (sx<0)    ? 0    : sx;          /* force sx into 0 to maxx */
sx = (sx>maxx) ? maxx : sx;
sy = (sy<0)    ? 0    : sy;          /* force sy into 0 to maxy */
sy = (sy>maxy) ? maxy : sy;

code = 0x1b;
code |= sx << 8;
outport(0x3d4, code);          /* set the x-start position */

ex = (xscale + 1) * (Window_Width - 1) + sx;
code = 0x1c;
code |= ex << 8;
outport(0x3d4, code);

code = 0x1d;
code |= (sy & 0xff) << 8;
outport(0x3d4, code);

code = 0x1f;
code |= ((sy & 0xff00) << 3);
outport(0x3d4, code);

ey = (yscale + 1) * (Window_Height) * Char_Height + sy - 1;
code = 0x1e;
code |= (ey & 0xff) << 8;
outport(0x3d4, code);

outportb(0x3d4, 0x1f);
code = inportb(0x3d5);
code &= 0xf8;
code |= (ey & 0xff00) >> 8;
outportb(0x3d4, 0x1f);
outportb(0x3d5, (char) (code & 0xff));

code = 0x20;
outport(0x3d4, code);

code ++;
outport(0x3d4, code);

outportb(0x3d4, 0x23);
code = inportb(0x3d5);
code &= 0xfb;
```

```
        outportb(0x3d4, 0x23);
        outportb(0x3d5, (char) (code & 0xff));
    }
```

到目前为止，我们所介绍的这些驻留程序设计技术，已足以应付一般中小规模驻留程序了。以下几节我们将运用这些技术设计两个一般程序设计常常用到的驻留程序工具——驻留打印图形工具以及驻留绘图小精灵。

7.6 单色驻留打印图形程序

本节所要介绍的这个程序，驻留部分的架构基本上是沿袭上一节的 ZOOMTSR.C 程序而来，印图部分则自第六章的印图程序 MPTRSCR.C 及 CPT RSCR.C 修改而来，所以不再对驻留程序及打印机控制部分多加说明，仅就设计驻留打印图形程序时所需注意到的关键部分提出讨论。先来看看本程序除了驻留程序所需的函数及变量外，还定义了哪些函数及变量：

```
#define RShift    0x0001
#define LShift    0x0002
#define Ctrl      0x0004
#define SpecialKey (* ((unsigned int far *) 0x00400017))

void p__putc(char);
void Hercules__printscr(int, int, int, int,int);
void HerculesMapping(void);
int  Getpixel(unsigned, unsigned, int);

extern   void far     *__heapbase;
extern   unsigned      __heaplen = 0x800;
extern   unsigned      __stklen  = 0x800;

static   unsigned      HotKey0 = RShift+Ctrl;
static   unsigned      HotKey1 = LShift+Ctrl;
static   unsigned      address[348];
```

单色绘图模式一般都会有两个绘图页，因此本程序提供了两个热键来引导打印图形的功能，分别是 Hotkey0（按右边的 Shift 键和 Ctrl 键）打印第 0 页以及 Hotkey1（按左边的 Shift 键和 Ctrl 键）打印第 1 页图形。

函数 Hercules__printscr，P__putc，HerculesMapping 及 Getpixel 都是和打印相关的函数，将于后面讨论。另外，在这里我们分别指定给__heaplen 及__stklen 各 2KB 的空间以节省内存的使用，由这两个变量可以使这个驻留程序只占 11KB 左右的内存。

接下来以中断函数 new1c 为例，说明在中断中如何去激活打印图形函数 Hercules__printscr:

```
void interrupt new1c()
```

```
{
    (*old1c) ()

    disable();
    if(TsrBusy) return;
    TsrBusy++;
    enable();

    OldSS=_SS;
    OldSP=_SP;
    _SS =TsrSS;
    _SP =TsrSP;

    if((SpecialKey & HotKey0)= =HotKey0)
            Hercules_printscr(0,0,719,347,0);
    if((SpecialKey & HotKey1)= =HotKey1)
            Hercules_printscr(0,0,719,347,1);

    _SS = OldSS;
    _SP = OldSP;
    TsrBusy--;
}
```

new1c 中主要是做按键检查的任务:

1. 若是用户按下Hotkey0，则引导Herculer_printscr函数打印第0页图形。
2. 若是按下 Hotkey1 则打印第 1 页。

因为 Hercules_printscr 函数很简单，故无需判断 DOS 的忙碌标志，其它 new09 及 new28 的内容也和此函数相同。我们比较感兴趣的是印图的函数部分做了哪些改变。先来看看 Hercules_printscr 函数:

```
void Hercules_printscr(int sx, int sy, int ex, int ey, int pgae)
{
    int deltax;
    int n1, n2;
    int i, j, k;
    char data;

    HerculesMappint();
    p_putc(ESC);
    p_putc('3');
    p_putc(24);
    deltax=ex - sx + 1;
    n1=deltax % 256;
    n2=deltax / 256;
```

```
    for(i=sy; i<=ey; i+=8)
        p_putc(ESC);
        p_putc('L');
        p_putc(n1);
        p_putc(n2);
        for(j=sx; j<=ex; j++)
        {
            data=0;
            for(k=0; k<8; k++)
                if((i+k)<=ey)
                    if(Getpixel(j, i+k, page))
                        data |=(1<<(7-k));
            p_putc('\n');
            p_putc('\r');
    }
    p_putc(ESC);
    p_putc('2');
}
```

这个函数和以前所介绍的非驻留版本有三项差异，分别是：

1. 调用时多了一项页数选择 page。
2. 在函数之前调用了 Hercules_Mapping()函数。
3. 使用自行编写的读点函数 Getpixel()。

其中，页数选择是为了因打印不同页别的需要而加入的，而 Hercules_Mapping 函数则是产生 Getpixel 函数查表所需的表格 address[348]，它的内容如下所示：

```
void HerculesMapping()
{
    int i;
    for(i=0; i<348; i++)
      address[i] = 8192 * (i%4) + 90 * (i/4);
}
```

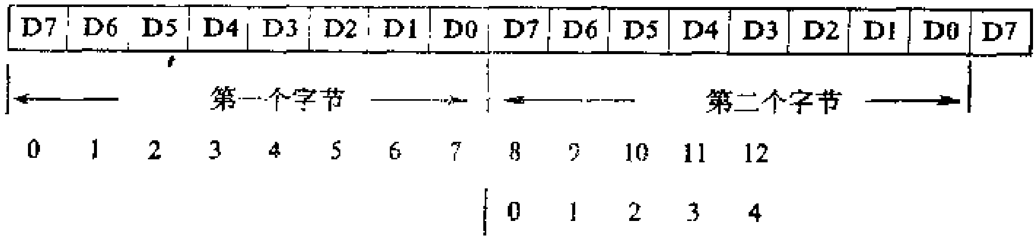
我们知道 Hercules 绘图卡的显示内存和扫描线并不呈线性对应，所以先用此函数以每条扫描线为单位，产生一个表格 address，以利程序对其显示内存的操作。这个函数其实只要运行过一次即可，所以你也可以放到 main 函数中去运行。除此以外就是 Getpixel 函数了：

```
int Getpixel(unsigned x, unsigned y, int page)
{
    unsigned bitsno;
    unsigned char data;

    bitsno = 7-(x%8);
    data   = peekb( (page) ? 0xb800 :
```

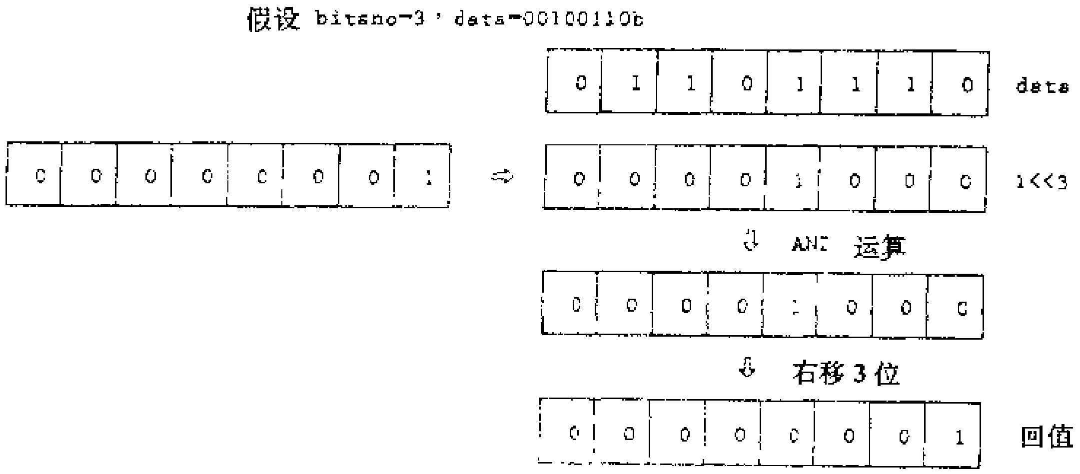
```
                0xb000, address[y]+x / 8);
    return( (data & (1<tsnol) ) >> bitsno);
}
```

各位一定还记得，在非驻留版本的印图程序中是使用 Turbo C 所提供的绘图函数 getpixel()来完成读点的任务。为什么在驻留程序版本中要特别再编写一个读点的函数呢？因为在 Turbo C 中要使用任何绘图函数之前，都必须先使用 initgraph()函数定义进入绘图模式，然后 Turbo C 就会做一连串的各项绘图函数所需的初始化任务，当然，这也包括清除屏幕的操作。试想，我们使用驻留程序打印图形，就是为了捕捉前台程序的任一个画面，而当我们一激活打印图形程序时，画面就立刻被清除干净，那么这个驻留印图程序就没有意义了。所以在驻留程序中应尽量避免使用 initgraph()，而有关的绘图函数就必须自行设计了。

Getpixel()的第一个步骤是算出要读取像素点的 X 坐标是位于字节的哪一个位上，由于一个字节共有 8 个位，所以只要对 X 坐标取 8 的余数，再用 7 减即可。举个例子，假设 X 坐标值为 12，则取 8 的余数是 4。亦即下图第二个字节左边数来第 4 个位，再以 7 减 4，正好得到 3，即为 D3，所以 X 坐标 12，其 bitsno 等于 3。



有了 bitsno 后，再来就是取得该字节的数据。先以 address[y]+X / 8 算出该字节在内存中的地址，再以 keepb 将该字节取出即可。前文说过，address 所记录的是每一条扫描线的起始地址，把起始地址加上 X 坐标整除 8 后的数，就是该字节在内存中的实际地址。接下来就是要取得该字节上第 bitsno 个位的值是 0 或是 1，将其返回。运算方法如下：



最后一个要注意的地方是 p putc 函数中不能再使用 fprintf()函数了，必须改为中断 17h，程序如下：

```
void p_putc(char ch)
{
    r.h.ah = 0;
    r.h.al = ch;
    r.h.dx = 0;
    int86(0x17, &r, &r);
}
```

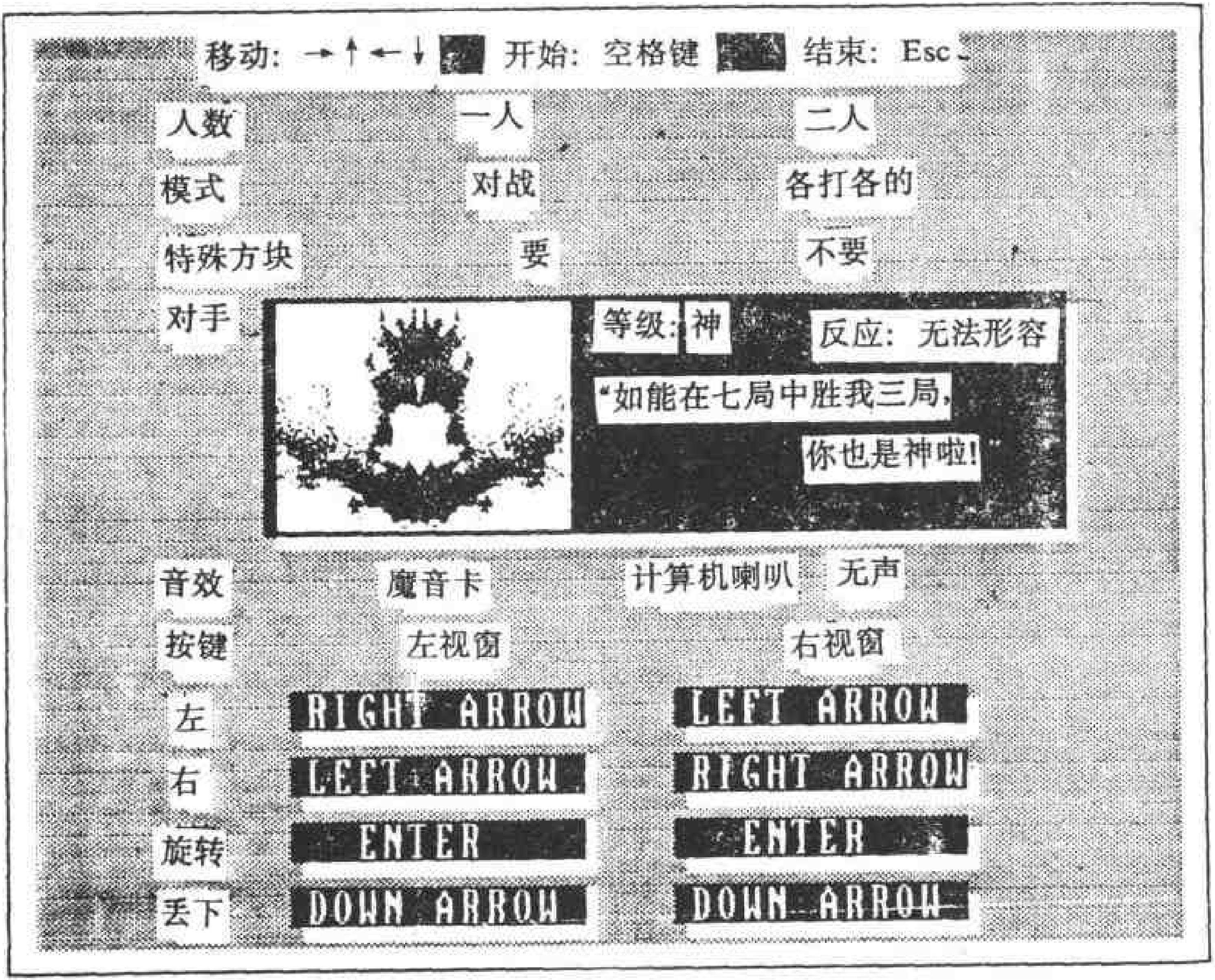
如果再使用 fprintf()函数，可能会造成 DOS 的重入而错误，应当注意。

现在就来看看这个程序的打印效果：（以电动玩具"决战俄罗斯"为例，打印机是 Canon BJ−10e）

图 7−7　印图程序 TSRPM.C 的运行实例

以下就是程序 TSRPM.C 的完整清单，供各位读者参考：

```
/* TSRPM.C */

#include <stdio.h>
#include <dos.h>
#include <bios.h>
```


移动: →↑←↓
开始: 空格键
结束: Esc
人数
一人
二人
模式
对战
各打各的
特殊方块
要
不要
对手
等级: 神
反应: 无法形容
"如能在七局中胜我三局,
你也是神啦!
音效
魔音卡
计算机喇叭
无声
按键
左视窗
右视窗
左
RIGHT ARROW
LEFT ARROW
右
LEFT ARROW
RIGHT ARROW
旋转
ENTER
ENTER
丢下
DOWN ARROW
DOWN ARROW

```
void p__putc(char ch)
{
    r.h.ah = 0;
    r.h.al = ch;
    r.h.dx = 0;
    int86(0x17, &r, &r);
}
```

如果再使用 fprintf()函数，可能会造成 DOS 的重入而错误，应当注意。

现在就来看看这个程序的打印效果：（以电动玩具"决战俄罗斯"为例，打印机是 Canon BJ-10e）

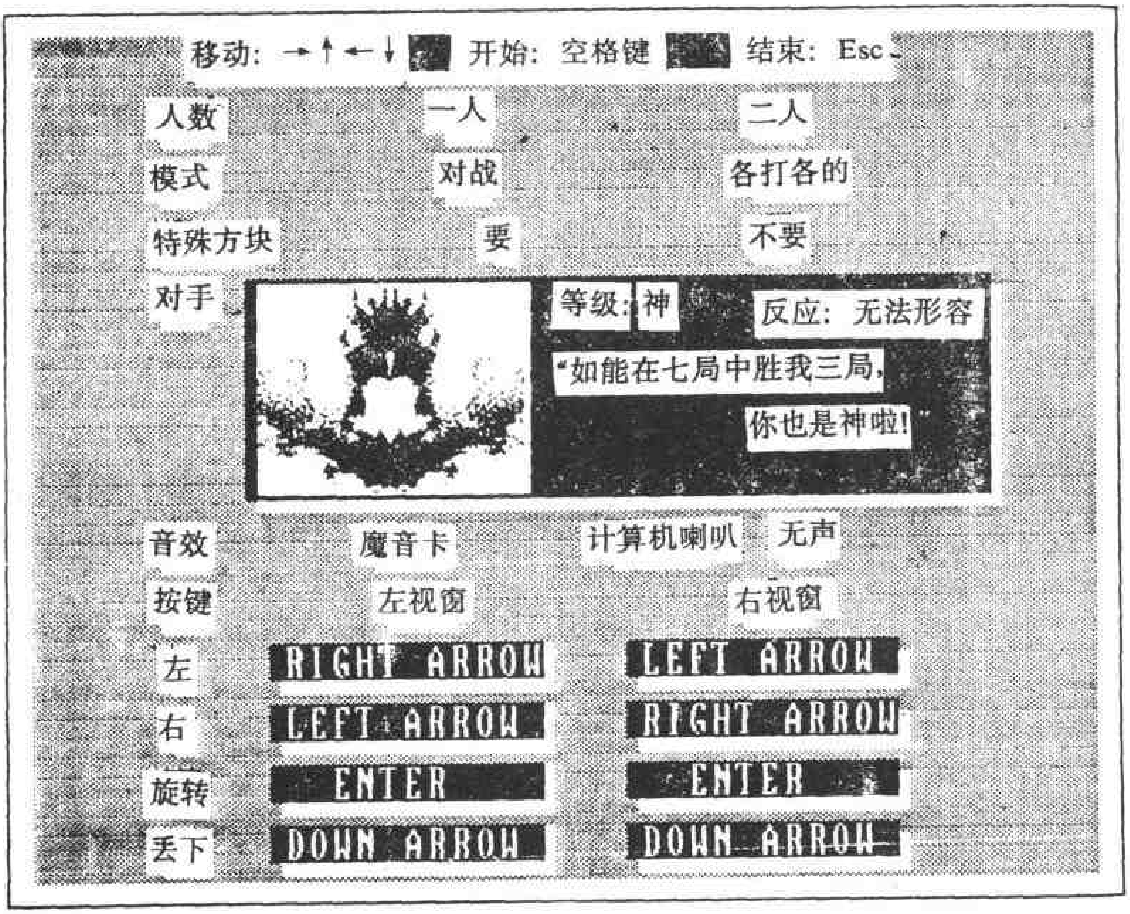


图 7–7 印图程序 TSRPM.C 的运行实例

以下就是程序 TSRPM.C 的完整清单，供各位读者参考：

```
/* TSRPM.C */

#include <stdio.h>
#include <dos.h>
#include <bios.h>
```

```
#include <stdlib.h>

#define ESC      27

#define RShift  0x0001
#define LShift  0x0002
#define Ctrl    0x0004
#define SpecialKey (*((unsigned int far *) 0x00400017))

void interrupt new09();
void interrupt (*old09)();
void interrupt new1c();
void interrupt (*old1c)();
void interrupt new28();
void interrupt (*old28)();
void interrupt (*old60)();
void interrupt new61();
void interrupt (*old61)();
int  TsrInMem(void);
void p_putc(char);
void Hercules_printscr(int, int, int, int, int);
void HerculesMapping(void);
int  Getpixel(unsigned, unsigned, int);

extern void far *_heapbase;
extern unsigned _heaplen=0x800;
extern unsigned _stklen =0x800;

static unsigned HotKey0 = RShift + Ctrl;
static unsigned HotKey1 = LShift + Ctrl;
static unsigned TsrSS, TsrSP, TsrPSP;
static unsigned OldSS, OldSP;
static unsigned TsrBusy=0;
static char far *DosBusy;
static unsigned address[348];
union REGS r;
struct SREGS sr;
unsigned char id[4] = {0x89, 0xab, 0xed, 0xef};

main(argc, argv)
int argc;
char *argv[];
{
  delay(0);
```

```
        if(argc > 1)
          if(!strnicmp(argv[1], "remove", 5)) {
                if(!TsrInMem()) {
                    puts("There is no PrintGraph Utility [Mono] in memory!");
                    exit(0);
             }
             else {
                  __int__(0x61);
                  puts("PrintGraph Utility [Mono] was already removed!");
                    exit(0);
             }
        }

        if(TsrInMem()) {
            puts("PrintGraph Utility [Mono] was already residented!");
        exit(0);
        }

        old60 = getvect(0x60);
        movedata(FP_SEG(id), FP_OFF(id), 0, 0x180, 4);
      disable();
      old09 = getvect(0x09);
      setvect(0x09, new09);
      old1c = getvect(0x1c);
      setvect(0x1c, new1c);
      old28 = getvect(0x28);
      setvect(0x28, new28);
      old61 = getvect(0x61);
      setvect(0x61, new61);
      enable();

      TsrSS = _SS;
      TsrSP = _SP;
      TsrPSP = _psp;
      r.h.ah = 0x34;
      intdosx(&r, &r, &sr);
      DosBusy = MK_FP(sr.es, r.x.bx);

      puts("[[[ * PrintGraph Utility [Mono] * ]]] install...");
      keep(0, FP_SEG(_heapbase)-_psp);
      puts("Can't install TSR program!");
}

void interrupt new1c()
```

```
 {
     ( * old1c)();

     disable();
     if(TsrBusy) return;
     TsrBusy++;
     enable();

     OldSS = _SS;
     OldSP = _SP;
     _SS = TsrSS;
     _SP = TsrSP;

     if((SpecialKey & HotKey0)==HotKey0)
             Hercules_printscr(0,0,719,347,0);
     if((SpecialKey & HotKey1)==HotKey1)
             Hercules_printscr(0,0,719,347,1);

     _SS = OldSS;
     _SP = OldSP;
     TsrBusy--;
}

void interrupt new09()
{
     ( * old09)();

     disable();
     if(TsrBusy) return;
     TsrBusy++;
     enable();

     OldSS = _SS;
     OldSP = _SP;
     _SS = TsrSS;
     _SP = TsrSP;

     if( (SpecialKey & HotKey0)==HotKey0 )
             Hercules_printscr(0,0,719,347,0);
     if( (SpecialKey & HotKey1)==HotKey1 )
             Hercules_printscr(0,0,719,347,1);

     _SS = OldSS;
     _SP = OldSP;
```

```
        TsrBusy--;
    }

    void interrupt new28()
    {
        ( * old28)();

        disable();
        if(TsrBusy) return;
        TsrBusy++;
        enable();

        OldSS = _SS;
        OldSP = _SP;
        _SS = TsrSS;
        _SP = TsrSP;

           if((SpecialKey & HotKey0) == HotKey0)
                Hercules_printscr(0,0,719,347,0);
        if((SpecialKey & HotKey1) == HotKey1)
                Hercules_printscr(0,0,719,347,1);

        _SS = OldSS;
        _SP = OldSP;
        TsrBusy--;
    }

    void interrupt new61()
    {
        disable();
        setvect(0x09, old09);
        setvect(0x1c, old1c);
        setvect(0x28, old28);
        setvect(0x60, old60);
        setvect(0x61, old61);
        _ES = TsrPSP;
        _AH = 0x49;
        __int__(0x21);
        _ES = peek(TsrPSP, 0x2c);
        _AH = 0x49;
        __int__(0x21);
        enable();
    }
```

```
int TsrInMem()
{
        unsigned char Int60[4];

        movedata(0, 0x180, FP_SEG(Int60), FP_OFF(Int60), 4);
        if(!strncmp(Int60, id, 4)) return 1;
        else return 0;
}

void p_putc(char ch)
{
        r.h.ah = 0;
        r.h.al = ch;
        r.x.dx = 0;
        int86(0x17, &r, &r);
}

void Hercules_printscr(int sx,int sy,int ex,int ey,int page)
{
        int deltax;
        int n1, n2;
        int i, j, k;
        char data;

        HerculesMapping();
        p_putc(ESC);
        p_putc('3');
        p_putc(24);
        deltax = ex - sx + 1;
        n1 = deltax % 256;
        n2 = deltax / 256;
        for(i=sy; i<=ey; i+=8)
             p_putc(ESC);
             p_putc('L');
             p_putc(n1);
             p_putc(n2);
             for(j=sx; j<=ex; j++)
        {
               data = 0;
               for(k=0; k<8; k++)
                    if((i+k)<=ey)
                       if(Getpixel(j, i+k, page))
                       data   = (1<<(7-k));
              p_putc(data);
```

```
            }
            p__putc('\n');
            p__putc('\r');
        }
        p__putc(ESC);
        p__putc('2');
}

int Getpixel(unsigned x, unsigned y, int page)
{
        unsigned bitsno;
        unsigned char data;

        bitsno   = 7-(x%8);
        data = peekb( (page) ? 0xb800 :
                    0xb000, address[y] + x / 8);
        return( ( data & (1<<bitsno) ) >> bitsno );
}

void HerculesMapping()
{
        int i;
        for(i=0; i<348; i++)
        address[i] = 8192 * (i%4) + 90 * (i/4);
}
```

7.7 彩色驻留式打印图形程序

上一节的 TSRPM.C 程序是以单色屏幕为例，若是要改成彩色打印图形的话，则有下面几项差异：

1. 16色绘图模式主要有两种分辨率，分别是640×350及640×480，在程序中要加以判断。

```
if(VideoMode= =0x10)        scanline=350;
else                        scanline=480;
if( (SpacialKey & HotKey0) = =HotKey0)
        EGAVGA__printscr(0, 0, 639, scanline-1, 0, 0);
if( (SpecialKey & HotKey1)= =HotKey1)
        EGAVGA__printscr(0, 0, 639, scanline-1, 0, 1);
```

2. 由于BIOS有支持彩色读点的功能，因此无需自行操作显示内存，当然也就不需要定位的函数了。

下面是彩色图形以单色打印的实例：（以电动玩具“三国志Ⅲ”为例，打印机是 Canon BJ—10e）。

图 7-8　彩色图形打印程序 TSRPC.C 的运行实例

```
int Getpixel(unsigned x, unsigned y, int page)
{
    r.h.ah = 0x0d;
    r.h.bh = page;
    r.x.cx = x;
    r.x.dx = y;
    int86(0x10, &r, &r);
    return(r.h.al);
}
```

以下是完整的程序清单：(TSRPC.C)

```
/* TSRPC.C */

    #include <stdio.h>
    #include <dos.h>
    #include <bios.h>
    #include <stdlib.h>

    #define ESC      27

    #define RShift  0x0001
    #define LShift  0x0002
    #define Ctrl     0x0004
    #define SpecialKey (*((unsigned int far *) 0x00400017))
    #define VideoMode  (*((unsigned char far *)0x00400049))
```

1. Start new game
2. Load data
3. Quit game
Select which (1-3)?

图 7-8 彩色图形打印程序 TSRPC.C 的运行实例

```
int Getpixel(unsigned x, unsigned y, int page)
{
    r.h.ah = 0x0d;
    r.h.bh = page;
    r.x.cx = x;
    r.x.dx = y;
    int86(0x10, &r, &r);
    return(r.h.al);
}
```

以下是完整的程序清单：(TSRPC.C)

```
/* TSRPC.C */

    #include <stdio.h>
    #include <dos.h>
    #include <bios.h>
    #include <stdlib.h>

    #define ESC      27

    #define RShift  0x0001
    #define LShift  0x0002
    #define Ctrl    0x0004
    #define SpecialKey (*((unsigned int far *) 0x00400017))
    #define VideoMode  (*((unsigned char far *)0x00400049))
```

```
void interrupt new09();
void interrupt ( * old09)();
void interrupt new1c();
void interrupt ( * old1c)();
void interrupt new28();
void interrupt ( * old28)();
void interrupt ( * old60)();
void interrupt new61();
void interrupt ( * old61)();
int  TsrInMem(void);
void p_putc(char);
void EGAVGA_printscr(int, int, int, int, int, int);
int  Getpixel(unsigned, unsigned, int);

extern void far  * _heapbase;
extern unsigned _heaplen = 0x800;
extern unsigned _stklen  = 0x800;

static unsigned HotKey0  =  RShift + Ctrl;
static unsigned HotKey1  =  LShift + Ctrl;
static unsigned TsrSS, TsrSP, TsrPSP;
static unsigned OldSS, OldSP;
static unsigned TsrBusy = 0;
static char far  * DosBusy;
static unsigned scanline;
union REGS r;
struct SREGS sr;
unsigned char id[4] = {0x89, 0xab, 0xcd, 0xef};

main(argc, argv)
int argc;
char  * argv[];
{
        delay(0);
        if(argc > 1)
        if(!strnicmp(argv[1], "remove", 5)) {
        if(!TsrInMem()) {
                puts("There is no PrintGraph Utility [Color] in memory!");
                exit(0);
        }
        else {
                __int__(0x61);
                puts("PrintGraph Utility [Color] was already removed!");
                exit(0);
```

```
        }
        }

        if(TsrInMem()) {
        puts("PrintGraph Utility [Color] was already residented!");
        exit(0);
        }

        old60 = getvect(0x60);
        movedata(FP_SEG(id), FP_OFF(id), 0, 0x180, 4);
        disable();
        old09 = getvect(0x09);
        setvect(0x09, new09);
        old1c = getvect(0x1c);
        setvect(0x1c, new1c);
        old28 = getvect(0x28);
        setvect(0x28, new28);
        old61 = getvect(0x61);
        setvect(0x61, new61);
        enable();

        TsrSS = _SS;
        TsrSP = _SP;
        TsrPSP = _psp;
        r.h.ah = 0x34;
        intdosx(&r, &r, &sr);
        DosBusy = MK_FP(sr.es, r.x.bx);

        puts("[[[ * PrintGraph Utility [Color] * ]]] install...");
        keep(0, FP_SEG(_heapbase)-_psp);
        puts("Can't install TSR program!");
}

void interrupt new1c()
{
            ( * old1c)();

        disable();
            if(TsrBusy) return;
            TsrBusy++;
            enable();

            OldSS = _SS;
            OldSP = _SP;
```

```
        _SS = TsrSS;
        _SP = TsrSP;

        if(VideoMode==0x10) scanline=350;
        else                scanline=480;
        if((SpecialKey & HotKey0)==HotKey0)
            EGAVGA_printscr(0,0,639,scanline-1,0,0);
        if((SpecialKey & HotKey1)==HotKey1)
            EGAVGA_printscr(0,0,639,scanline-1,0,1);

        _SS = OldSS;
        _SP = OldSP;
        TsrBusy--;
}

    void interrupt new09()
{
        (*old09)();

        disable();
        if(TsrBusy) return;
        TsrBusy++;
        enable();

        OldSS = _SS;
        OldSP = _SP;
        _SS = TsrSS;
        _SP = TsrSP;

        if(VideoMode==0x10) scanline=350;
        else                scanline=480;
        if((SpecialKey & HotKey0)==HotKey0)
            EGAVGA_printscr(0,0,639,scanline-1,0,0);
        if((SpecialKey & HotKey1)==HotKey1)
            EGAVGA_printscr(0,0,639,scanline-1,0,1);

        _SS = OldSS;
        _SP = OldSP;
        TsrBusy--;
}

void interrupt new28()
{
        (*old28)();
```

```
        disable();
        if(TsrBusy) return;
        TsrBusy++;
        enable();

        OldSS = _SS;
        OldSP = _SP;
        _SS = TsrSS;
        _SP = TsrSP;

        if(VideoMode==0x10) scanline=350;
        else                scanline=480;
        if((SpecialKey & HotKey0)==HotKey0)
             EGAVGA_printscr(0,0,639,scanline-1,0,0);
        if((SpecialKey & HotKey1)==HotKey1)
                 EGAVGA_printscr(0,0,639,scanline-1,0,1);

        _SS = OldSS;
        _SP = OldSP;
        TsrBusy--;
}

void interrupt new61()
{
        disable();
        setvect(0x09, old09);
        setvect(0x1c, old1c);
        setvect(0x28, old28);
        setvect(0x60, old60);
        setvect(0x61, old61);
        _ES = TsrPSP;
        _AH = 0x49;
        __int__(0x21);
        _ES = peek(TsrPSP, 0x2c);
        _AH = 0x49;
        __int__(0x21);
        enable();
}

int TsrInMem()
{
        unsigned char Int60[4];
```

```
        movedata(0, 0x180, FP_SEG(Int60), FP_OFF(Int60), 4);
        if(!strncmp(Int60, id, 4)) return 1;
        else return 0;
}

void p_putc(char ch)
{
        r.h.ah = 0;
        r.h.al = ch;
        r.x.dx = 0;
        int86(0x17, &r, &r);
}

void EGAVGA_printscr(int sx, int sy, int ex, int ey, int bkcolor, int page)
{
        int deltax;
        int n1, n2;
        int i, j, k;
        char data;

        p_putc(ESC);
        p_putc('3');
        p_putc(24);
        deltax = ex - sx + 1;
        n1 = deltax % 256;
        n2 = deltax / 256;
        for(i=sy; i<=ey; i+=8)        {
                p_putc(ESC);
                p_putc('L');
                p_putc(n1);
                p_putc(n2);
                for(j=sx; j<=ex; j++)        {
                data = 0;
                for(k=0; k<8; k++)
                        if((i+k)<=ey)
                        if(Getpixel(j, i+k, page)!=bkcolor)
                        data |= (1<<(7-k));
                p_putc(data);
                }
                p_putc('\n');
                p_putc('\r');
        }
        p_putc(ESC);
        p_putc('2');
```

```
}

int Getpixel(unsigned x, unsigned y, int page)
{
    r.h.ah = 0x0d;
    r.h.bh = page;
    r.x.cx = x;
    r.x.dx = y;
    int86(0x10, &r, &r);
    return(r.h.al);
}
```

7.8 驻留绘图程序设计——绘图小精灵

本节将要介绍一个完整的驻留程序设计实例——绘图小精灵 ELF.C。这个程序主要的结构来自于第六章的绘图程序 MGETPIC.C，它的功能很强，请参阅表 7-8。

表 7-8 绘图小精灵的功能

功　　能	按　　键
决定区块大小	箭头键及空格键
将区块存储成.TPC 文件	Alt-S
将图形载入缓冲区	Alt-G
显示缓冲区内的图形	Alt-D
显示目前区块的相关信息	Alt-I
将区块内图形反相	Alt-R
.TPC 图形文件载入	Alt-L
将区块变亮	Alt-T
将区块变暗	Alt-K
将区块图形镜射	Alt-M
将区块图形上下颠倒	Alt-F
打印区块	Alt-P
离开	ESC 键

和 MGETPIC.C 比较起来，许多函数的写法已经有相当大的改变，最主要的原因乃是在于驻留程序中无法使用 initgraph()函数定义进入 Turbo C 的绘图模式（该函数会清除屏幕），以致于所有 Turbo C 的绘图函数包括 getpixel()，putpixel()，getimage()，putimage()……等都无法在程序中运用，只有自行设计。所以要编写驻留程序的绘图程序，得先从设计基本绘图函数开始。我们先从写点函数 putpixel()开始介绍：

```
void Putpixel(unsigned x, unsigned y, int color, int page)
{
```

```
    unsigned bitsno;
    unsigned char data;
    bitsno = 7-(x%8);
    data = peekb((page) ? 0xb800 : 0xb000, address[y]+x / 8);
    if(color)
      pokeb((page) ? 0xb800 : 0xb000, address[y]+x / 8, data | (1<tsno));
    else
      pokeb((page) ? 0xb800 : 0xb000, address[y]+x / 8,  data & (1<tsno));
}
```

Putpixel()函数的原理基本上和 Getpixel()函数相似，首先仍然先算出该点是位于某字节的第 bitsno 位上，接着到显示内存取出该字节，再将颜色值 color（0 或 1）填入该位即可。请看下图的讲解:

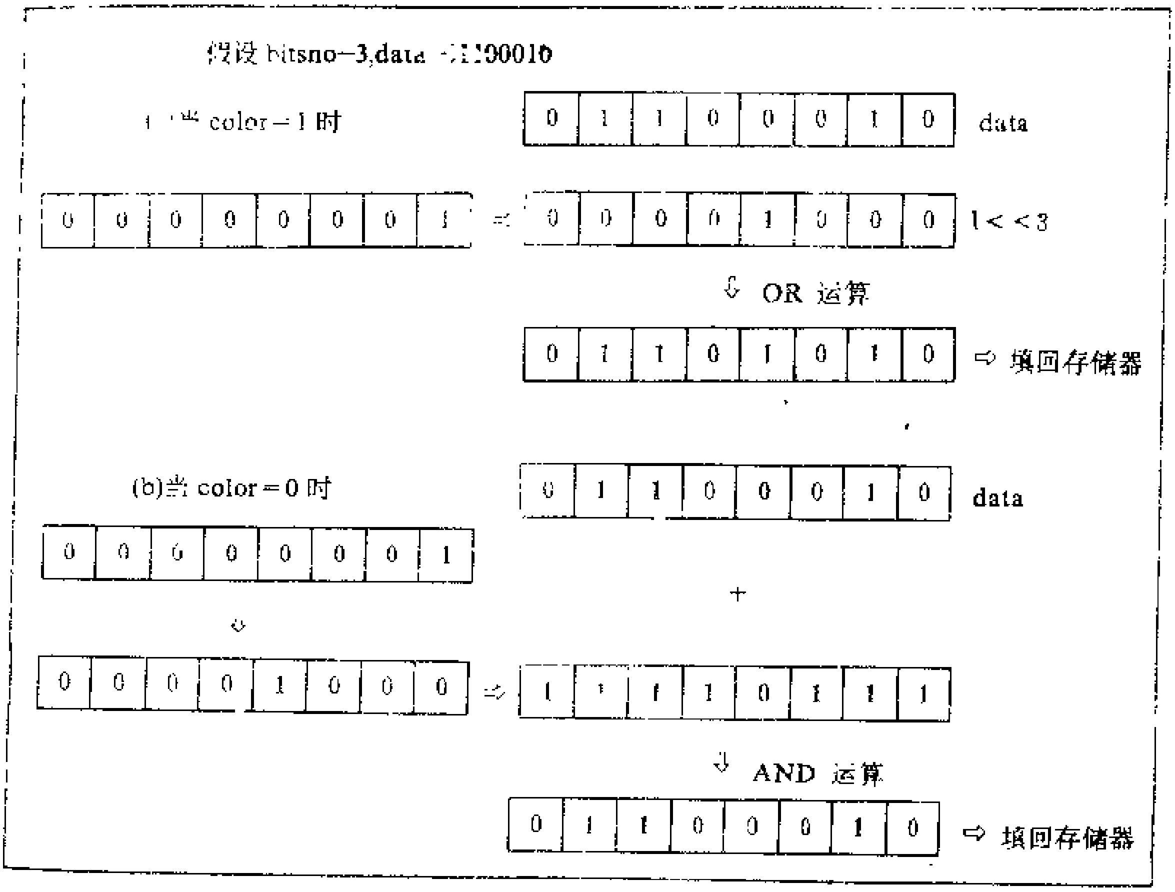


图 7-9 Putpixel()函数的原理

至于读点函数 Getpixel()已经在上一节介绍过了，不再重述。有了这两个函数，再编写其它驻留程序所需的绘图函数就没有那么困难了。下面是画水平线和垂直线例子:

```
void VLine(int x, int sy , int sy, int ey, int color, int page)
{
    int y;
```

```
    if(sy>ey)  { y=sy; sy=ey; ey=y; }
    for(y=sy; y<=ey;y++)
      Putpixel(x, y, color, page);
}
void HLine(int y, int sx, int ex, int color, int page)
{
    int x;

    if(sx>ex)  { x=sx; sx=ex; ex=x; }
    for(x=sx; x<=ex; x++)
      Putpixel(x, y, color, page);
}
```

此外，绘图程序中的方框通常为了防止破坏到背景，在画框时都是以 XOR 的方式画在屏幕上，所以还需要再设计两个以 XOR 运算方式画线的函数：

```
void VLineXor(int x, int sy, int ey, int page)
{
    int y;

    if(sy>ey)  { y=sy; sy=ey; ey=y; }
    for(y=sy; y<=ey;y++)
      if(Getpixel(x, y, page))
          Putpixel(x, y, 0, page);
      else
          Putpixel(x, y, 1, page);
}
void HLineXor(int y, int sx, int ex, int page)
{
    int x;

    if(sx>ex)  { x=sx; sx=ex; ex=x; }
    for(x=sx; x<=ex; x++)
      if(Getpixel(x, y, page))
          Putpixel(x, y, 0,page);
      else
          Putpixel(x, y, 1, page);
}
```

当然这两个函数在写点之前，并没有真正做 XOR 的逻辑运算，而是利用 XOR 的特性，当你要写一个 1 时，先检查该点目前状态是 1 或 0，若为 0，则写上 1；若是 1，则改写 0 上去即可。亦即当前台（要写入的值）及背景同时为 1 时则写 0，不同时为 1 时则写 1，因为这是一个画实心线的函数，所以前台恒为 1。

画点、读点及画线的函数有了，接着说明写字函数的功能：

```
void OutBitmapText(int x, int y, char *string, int mode)
{
    register int i, j;
    int len;

    HerculesMappint();
    len = strlen(string);
    for(i=0; i<len; i++)
        for(j=0; j<8; j++)
          if(mode==NORMAL)
            pokeb((ActivePage) ? 0xb800 : 0xb000, address[j+y]+i+x,
                  peekb(0xf000, 0xfa6e+string[i] * 8+j));
          else
            pokeb((ActivePage) ? 0xb800 : 0xb000, address[j+y]+i+x,
                  peekb(0xf000, 0xfa6e+string[i] * 8+j));
}
```

原来在 MGEPIC.C 中这就是一个直接存储器访问的函数，而且没有使用到任何 Turbo C 的绘图函数，所以在此唯一的改变只是加上一个页数选择的全局变量 ActivePage。这个变量记录着目前绘图程序所处理的页别。由于本函数主要作为和用户交谈之用，所以当然无条件输出到目前的作用页上。

绘图一定会对存储器操作并处理图像，下面三个图像处理的有关函数就显得相当重要：

```
unsigned Imagesize(int sx, int sy, int ex, int ey)
{
    unsigned dx, dy;

    dx = (ex-sx+1)/8+1;
    dy =  ey-sy+1;
    return((unsigned) dx * dy+6);
}
```

这个函数和 Turbo C 的 imagesize()函数相似，主要用来计算某一区块所占内存大小。

```
void Getmage(int sx, int sy, int ex, int ey, char *buf, int page)
{
    int x, y;
    int bits;

    HerculesMapping();
    *(buf++) = (ex-sx)%256;
```

```
        *(buf++) = (ex-sx)/256;
        *(buf++) = (ey-sy)% 256;
        *(buf++) = (ey-sy)/256;
        *buf = 0;
        bits = 7;
        for(y=sy; y<=ey; y++) {
            for(x=sx; x<=ex; x++) {
                if(Getpixel(x, y, page))
                    *buf |= 1 << bits;
                if(--bits<0) {
                    bits = 7;
                    *(++buf) = 0;
                }
            }
            *(++buf) = 0;
            bits = 7;
        }
}
```

Getimgae()函数和 Turbo C 的 getimage()函数相当，两者均是把(sx,sy)-(ex,ey)区块内的图像数据以相同的格式存储于指针变量 buf 处. 其原理主要是扫描屏幕上每一条扫描线（指定区块内）的图像数据，将它以每 8 点为单位收集成一个字节，存储至指定的缓冲区中. 以这种像素点扫描的方式，程序较易于理解，但速度较慢，读者不妨试试以字节的方式操作看看. 请看下面图解:

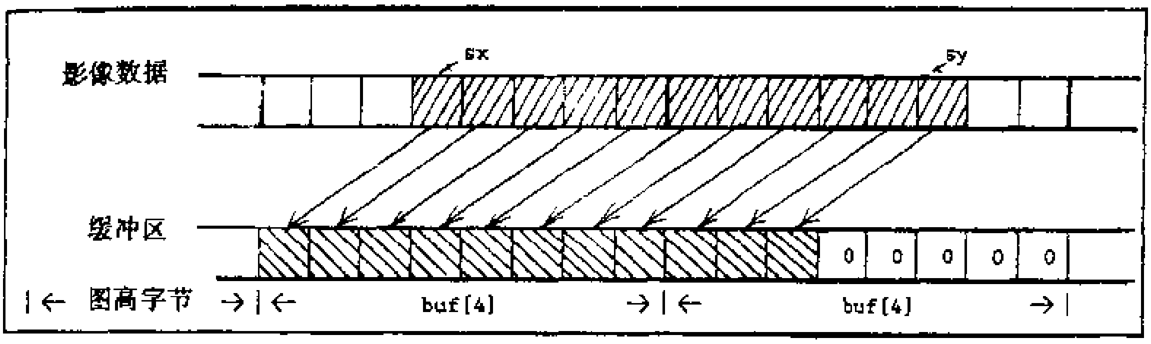


图 7-10 Getimage()函数的原理

```
void Putimage(int sx, int sy, char *buf, int page)
{
    int x, y, ex, ey;
    int bits;

    HerculesMapping();
    ex = ((unsigned)buf[1] & 0xff)*256+((unsigned)buf[0] & 0xff)+sx;
    ey = ((unsigned)buf[3] & 0xff)*256+((unsigned)buf[2] & 0xff)+sy;
```

```
        buf+=4;
        bits=y;

        for(y=sy; y<=ey; y++)    }
            for(x=sx; x<+ex; x++)  {
                if(((*buf)>>bits)&1)Putpixel(x, y, 1, page);
                else               Putpixel(x, y,0, page);
                if(—bits<0)
                    bits=7;
                    buf++;
                }
            }
            bits=7;
            buf++;
        }
    }
```

Putimage()函数则为 Getimage()函数的逆向操作。此函数只提供 1 种贴图模式 COPY_PUT，所以无需再另外指定，但是可别忘记指定负数。

以上就是绘图程序所必备的绘图函数。接下来我们再来看看原来 MGETPIC.C 的主要区块操作函数，在这里做了什么样的改变:

```
void RectXor(int sx, int sy, int dx, int dy)
{
    HLineXor(sy, sx, sx+dx, ActivePage);
    VLineXor(sx+dx, sy+1, sy+dy, ActivePage);
    HLinvXor(sy+dy, sx+dy-1, sx, ActivePage);
    VLineXor(sx, sy+dy, sy+1, ActivePage);
}
```

RectXor 函数中把原来的 line 函数改由 HLineXor 及 VLineXor 替代，较原来的函数简洁多了。

```
void Reverser(int sx, int sy, int dx, int dy)
{
    int x, y, ex, ey;

    ex=sx+dx;
    ey=sy+dy;
    for(x=sx; x<=ex; x++)
    for(y=sy; y<=ey;y++)
        if(Getpixel(x, y, ActivePage))
        Putpixel(x, y, 0, ActivePage);
         else
        Putpixel(x, y, 1, ActivePage);
}
```

Reverse()函数以 Getpixel()和 Putpixel()搭配运用，取代原来的 setwritemode (XOR_PUT) 和 line 函数。你也可以直接使用 VLineXor 函数。

```
void Light(int sx, int sy, int dx, int dy)
{
    register int i;

    for(i=sx; i<=sx=dx; i+=2)
       VLine(i, sy, sy+dy, 1, ActivePage);
}
void Dark(int sx, int sy, int dx, int dy)
{
    regsiter int i;

    for(i=sx; i<=sx+dx; i+=2)
       VLine(i, sy, sy+dy, 0,ActivePage);
}
```

Light 及 Dark 没有什么改变，只不过用 VLine 取代 line。

```
void Mirror(int sx, int sy, int dx, int dy)
{
    int ex, ey;
    int i, j;
    unsigned color;
    char *temp;

    ex=sx+dx;
    ey=sy+dy;
    for(i=sy; i<=ey; i++)
    for(j=sx; j<=ex; j++)
        color=Getpixel(j, i, ActviePage);
        Putpixel(ex-j+sx, i, color, abs(1-ActivePage));
      }
    temp=(char *) malloc(Imagesize(sx, sy, ex, ey));
    Getimage(sx, sy, ex, ey, temp,abs(1-ActivePage));
    Putimage(sx, sy, temp,ActivePage);
    free(temp);
}
void Filpflop(int sx, int sy, int dx, int dy)
{
    int ex, ey;
    int i, j;
    unsigned color;
```

```
    char * temp;

    ex=sx+dx;
    ey=sy+dy;
    for(i=sy; i<=ey; i++)
    for(j=sx; i<=ex; j++)    {
        color=Getpixel(j, i, ActivePage);
        Putpixel(j, ey-i+sy, color, abs(1-ActivePage));
    }
  temp=(char *)malloc(Imagesize(sx, sy, ex, ey));
  Getimage(sx, sy, ex, ey, temp,abs(1-ActivePage));
  free(temp);
}
```

由于我们特意把 Putimage()，Getimage()及 Imagesize()设计成和原来 Turbo C 所提供的函数兼容，所以 Mirror()及 Flipflop()二函数的内容无需做多大的调整，只要把函数名称换掉，再加上页数的指定就可以正常操作。同时这个函数中，我们以 ActivePage 变量来决定目前所作用的页别，而以另一页(abs(1-ActivePage))作为缓冲区存放暂时性的图像数据。

至于其它函数则大致雷同，就不在浪费篇幅，如有需要，请参考程序清单。现在我们再来看看在驻留程序部分做了什么变化。

```
void interrupt( * old62)();
viod interrupt new63();
void interrupt ( * old63)();
```

这个程序改以 INT 62h 做为识别记号中断矢量，而 INT 63h 为撤除驻留服务子程序，主要的目的是为了使本程序能和前面介绍的驻留程序共用。当然，最好的方法是另写一个函数来搜寻 60h~67h 之间没有用到的中断，这样驻留程序就会具有比较大的灵活性。

```
extern void far * _heapbase;
extern unsinged _heaplen=0x800;

static unsigned HotKey0=RShift+Alt;
static unsigned HotKey1=LShift+Alt;
static unsigned TsrSS, TsrSP, TsrPSP;
static unsigned TsrDTASet, TsrDTAOff;
static unsigned OldSS, OldSP,OldPSP;
static unsigned OldDTASet, OldDTAOff;
static unsigned TsrBusy=0;
static char far * DosBusy;
static unsigned ActivePage;
static unsigned address[348];
union REGS r;
```

```
struct SREGS sr;
unsigned char id[4]={0xaa, 0xaa, 0xaa, 0xaa};
```

在指定堆栈(heap)方面，程序指定保留 32KB(0x8000)的内存供堆栈使用。因为这个绘图程序很可能绘下全页的图形，而 heap 的功能主要就是供存储分配之用。在绘全页图形时自然需分配全页图形所需占的内存（约 30KB），所以分配 32KB 给堆使用有其必要性。

本程序的热键为避免和打印图形程序一样，改为左边的 Shift+Alt 键激活第 1 页绘图，而右边的 Shift+Alt 键激活第 0 页绘图。

此外，本程序在变量定义方面多出了 TsrDTASeg，TsrDTAoff……等，主要原因是绘图程序中有牵涉到驱动器的操作，所以在正式进入驻留程序之前，我们额外地再切换 PSP 及 DTA，以避免错误的发生。一般来说，只要驻留程序有使用到文件的操作，手续都会比较复杂。以下是 new1c 的程序码：

```
void interrupt new1c()
{
    (*old1c) ();

    disable());
    if(TsrBusy) return;
    TsrBusy++;
    enable();

    OldSS=_SS;
    OldSP=_SP;
    _SS=TsrSS;
    _SP=TsrSP;

    if( !(*DosBusy) ) {
        if( (SpecialKey & HotKey0)= =HotKey0) {
            ActivePage=0;
            mgetpic;
        }
        if( (SpecialKey & HotKey)= =HotKey1) {
            ActivePage=1;
            mgetpic;
        }
    }
    _SS=OldSS;
    _SP=OldSP;
    TsrBusy—;
}
```

程序中要特别注意的是正确地设置 ActivePage 变量，因为等进入 mgetpic 绘图程序后，所有的操作包括信息的显示、交谈方框……等，全部都以这个变量为准。同时，在进入 mgetpic()函数后，别忘了切换以下各项参量：

```
r.h.ah=0x2f;                    /* Save old  DTA */
intdosx(&r, &r, &sr);
OldDTASet=sr.es;
OldDTAOff=r.x.bx;
r.h.ah=0x1a;                    /* Set TSR's DTA  */
sr.ds=TsrDTASet;
r.h.dx=TsrDTAOff;
intdosx(&r, &r, &sr);

r.h.ah=0x51;
intdos(&r, &r):                 /* Save old PSP  */
OldPSP=r.x.bx;

r.h.ah=0x50;                    /* Set TSR's PSP  */
r.x.bx=TsrPSP;
intdos(&r, &r);
```

退出函数之前，也别忘了还原：

```
sr.ds=OldDTASeg;                 /* Restore Old DTA  */
r.x.dx=OldDTAOff;
r.h.ah=0x1a;
intdosx(&r, &r, &sr);

r.h.ah=0x50;                    /* Restore Old PSP  */
r.x.bx=OldPSP;
intdos(&r, &r);
```

程序编译完毕之后，我们只要在 DOS 下打入：

C:>elf

在按下 Enter 之后，绘图小精灵便驻留于内存中了 其引导方法如下：

绘图页第 0 页 ——按 ALT+右边的 Shift

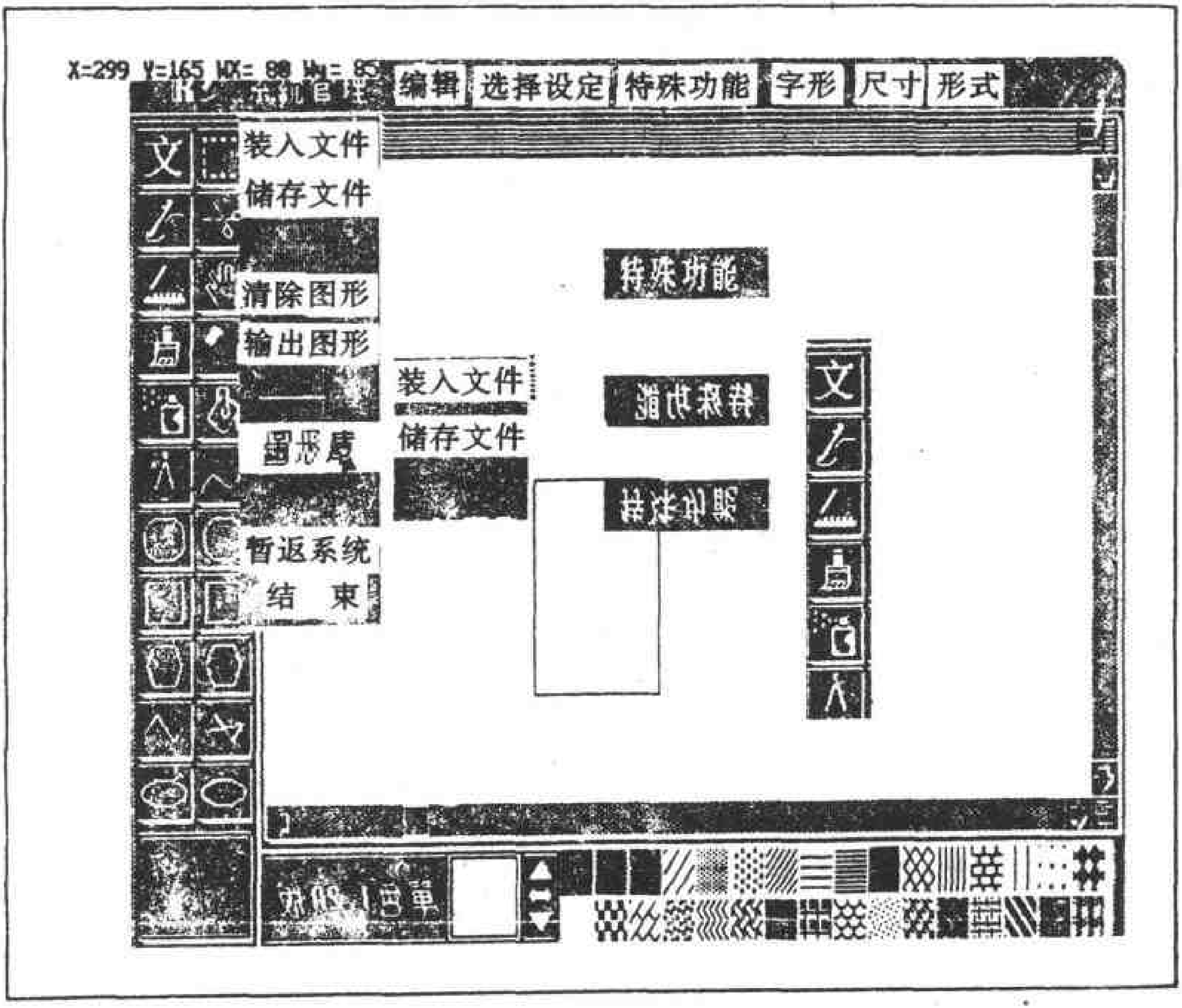
绘图页第 1 页 ——按 ALT+左边的 Shift

至于其它功能的操作方法，则请参考表 7-8。以下就是绘图小精灵程序 ELF.C 的运行实例以及完整的清单：

图 7-11 绘图小精灵程序 ELF.C 的运行实例

(以绘图软件 Rambow Paint 为例，打印机是 Canon BJ-10e)

```c
/ * ELF.C * /

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include "key.h"

#define TRUE     1
#define FALSE    0
#define ON       1
#define OFF      0
#define POST     0
```


编辑
选择设定
特殊功能
字形
尺寸
形式
装入文件
储存文件
清除图形
输出图形
暂返系统
结 束
装入文件
储存文件
特殊功能
文

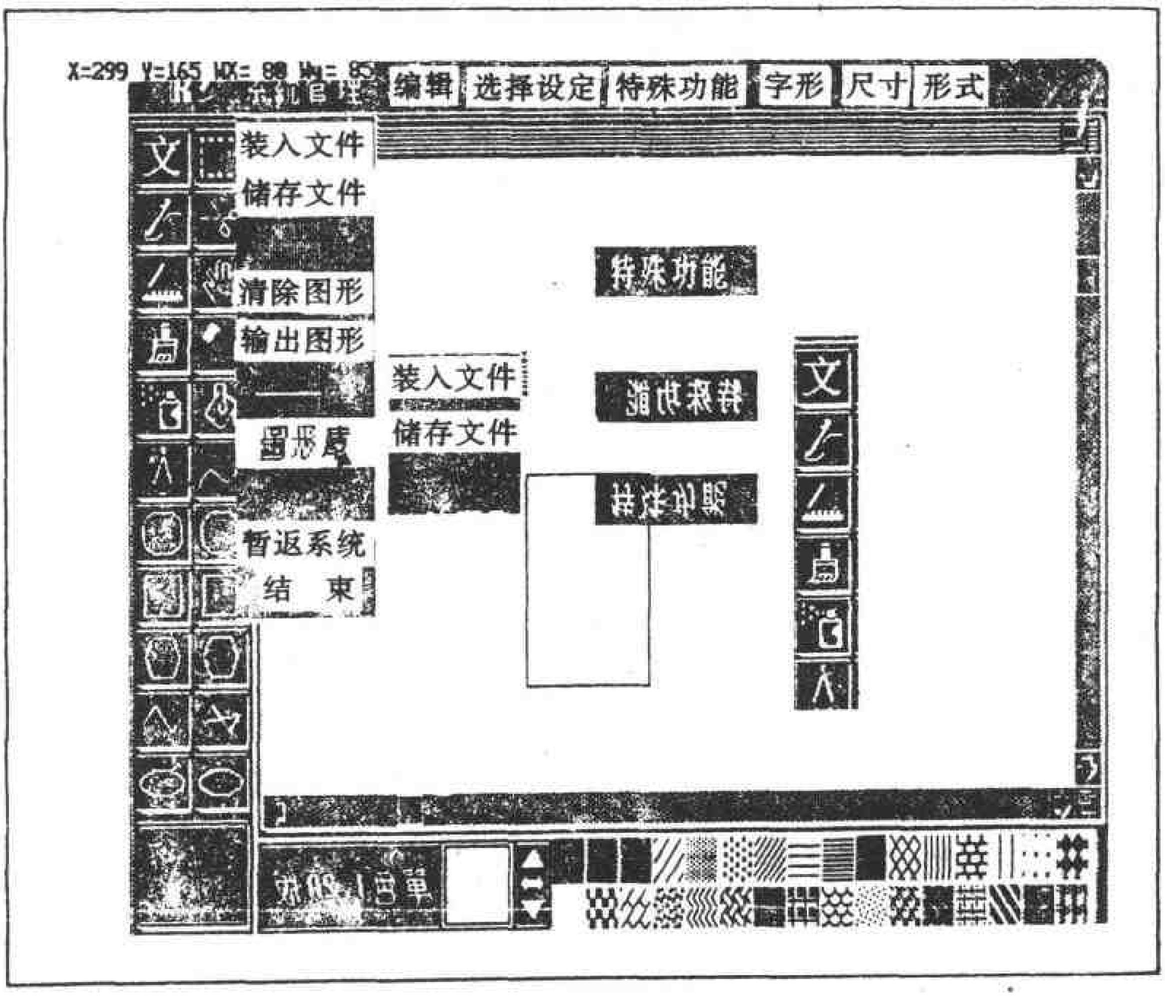


图 7-11　绘图小精灵程序 ELF.C 的运行实例

(以绘图软件 Rambow Paint 为例，打印机是 Canon BJ-10e)

```
/* ELF.C */

  #include <dos.h>
  #include <stdio.h>
  #include <stdlib.h>
  #include <conio.h>
  #include <alloc.h>
  #include "key.h"

  #define  TRUE     1
  #define  FALSE    0
  #define  ON       1
  #define  OFF      0
  #define  POST     0
```

```
#define  DELTA      1
#define  MaxX       720
#define  MaxY       348
#define  NORMAL     0
#define  REVERSE    1
#define  RShift     0x0001
#define  LShift     0x0002
#define  Alt        0x0008
#define  SpecialKey ( * ((unsigned int far  * ) 0x00400017))
void interrupt new09();
void interrupt ( * old09)();
void interrupt new1c();
void interrupt ( * old1c)();
void interrupt new28();
void interrupt ( * old28)();
void interrupt ( * old62)();
void interrupt new63();
void interrupt ( * old63)();
int  TsrInMem(void);

void mgetpic(void);
void OutBitmapText(int, int, char  * , int);
void Putpixel(unsigned, unsigned, int, int);
int  Getpixel(unsigned, unsigned, int);
void Putimage(int, int, char  * , int);
void Getimage(int, int, int, int, char  * , int);
unsigned Imagesize(int, int, int, int);
void VLine(int, int, int, int, int);
void HLine(int, int, int, int, int);
void VLineXor(int, int, int, int);
void HLineXor(int, int, int, int);
void HerculesMapping(void);
void RectXor(int, int, int, int);
void Light(int, int, int, int);
void Dark(int, int, int, int);
void Reverse(int, int, int, int);
void Getfilename(char  * );
void Saveimage(char  * , int, int, int, int);
void Loadimage(char  * , int, int);
void Flipflop(int, int, int, int);
void Mirror(int, int, int, int);
void Hercules_printscr(int, int, int, int, int);

extern void far * _heapbase;
```

```
extern unsigned _heaplen=0x8000;

static unsigned HotKey0 = RShift + Alt;
static unsigned HotKey1 = LShift + Alt;
static unsigned TsrSS, TsrSP, TsrPSP;
static unsigned TsrDTASeg, TsrDTAOff;
static unsigned OldSS, OldSP, OldPSP;
static unsigned OldDTASeg, OldDTAOff;
static unsigned TsrBusy=0;
static char far *DosBusy;
static unsigned ActivePage;
static unsigned address[348];
union REGS r;
struct SREGS sr;
unsigned char id[4] = {0xaa, 0xaa, 0xaa, 0xaa};

main(argc, argv)
int argc;
char *argv[];
{
  delay(0);
  if(argc>1)
  if(!strnicmp(argv[1], "remove", 5)) {
      if(!TsrInMem()) {
        puts("There is no GetPic elf [Mono] in memory!");
        exit(0);
      }
      else {
        __int__(0x63);
        puts("GetPic elf [Mono] was already removed!");
        exit(0);
      }
  }

  if(TsrInMem()) {
        puts("GetPic elf [Mono] was already residented!");
        exit(0);
  }

  old62 = getvect(0x62);
  movedata(FP_SEG(id), FP_OFF(id), 0, 0x188, 4);
  disable();
  old09 = getvect(0x09);
  setvect(0x09, new09);
```

```
        old1c = getvect(0x1c);
        setvect(0x1c, new1c);
        old28 = getvect(0x28);
        setvect(0x28, new28);
        old63 = getvect(0x63);
        setvect(0x63, new63);
        enable();

    TsrSS = _SS;
    TsrSP = _SP;
    TsrPSP = _psp;          /* Get PSP */

    r.h.ah = 0x34;          /* Get DosBusy */
    intdosx(&r, &r, &sr);
    DosBusy = MK_FP(sr.es, r.x.bx);

    r.h.ah = 0x2f;          /* Get DTA */
    intdosx(&r, &r, &sr);
    TsrDTASeg = sr.es;
    TsrDTAOff = r.x.bx;

    puts("[[[* GetPic elf [Mono] *]]] install...");
    keep(0, FP_SEG(_heapbase)-_psp);
    puts("Can't install TSR program!");
}

void interrupt new1c()
{
      (*old1c)();

      disable();
      if(TsrBusy) return;
      TsrBusy++;
      enable();

      OldSS = _SS;
      OldSP = _SP;
      _SS = TsrSS;
      _SP = TsrSP;

      if( !(*DosBusy) ) {
            if( (SpecialKey & HotKey0)==HotKey0 ) {
                  ActivePage=0; mgetpic();
            }
```

```
            if( (SpecialKey & HotKey1)==HotKey1)  {
                ActivePage=1; mgetpic();
            }
        }

        _SS = OldSS;
        _SP = OldSP;
        TsrBusy--;
}

void interrupt new09()
{
        ( * old09)();

        disable();
        if(TsrBusy) return;
        TsrBusy++;
        enable();

        OldSS = _SS;
        OldSP = _SP;
        _SS = TsrSS;
        _SP = TsrSP:

        if(!( * DosBusy))     {
            if( (SpecialKey & HotKey0)==HotKey0) {
                ActivePage=0;
                mgetpic();
            }
            if( (SpecialKey & HotKey1)==HotKey1) {
                ActivePage=1;
                mgetpic();
            }
        }

        _SS = OldSS;
        _SP = OldSP;
        TsrBusy--;
}

void interrupt new28()
{
        ( * old28)();
```

```
        disable();
        if(TsrBusy) return;
        TsrBusy++;
        enable();

        OldSS = _SS;
        OldSP = _SP;
        _SS = TsrSS;
        _SP = TsrSP;

        if((SpecialKey & HotKey0)==HotKey0)    {
              ActivePage=0;
              mgetpic();
        }
        if((SpecialKey & HotKey1)==HotKey1)    {
              ActivePage=1;
              mgetpic();
        }

        _SS = OldSS;
        _SP = OldSP;
        TsrBusy--;
    }

void interrupt new63()
{
       disable();
       setvect(0x09, old09);
       setvect(0x1c, old1c);
       setvect(0x28, old28);
       setvect(0x62, old62);
       setvect(0x63, old63);
       _ES = TsrPSP;
       _AH = 0x49;
       __int__(0x21);
       _ES = peek(TsrPSP, 0x2c);
       _AH = 0x49;
       __int__(0x21);
       enable();
    }

    int TsrInMem()
    {
      unsigned char Int62[4];
```

```
        movedata(0, 0x188, FP_SEG(Int62), FP_OFF(Int62), 4);
        if(!strncmp(Int62, id, 4)) return 1;
        else return 0;
    }

void mgetpic()
{
    int x=0, y=0, ox=0, oy=0;
    int deltax=10, deltay=10, odeltax=10, odeltay=10;
    int d=1;
    int key;
    int quit=FALSE;
    int IncDecFlag=POST;
    int DispInfo=OFF;
    int i;
    char *buf, *grpdata=NULL;


    r.h.ah = 0x2f;          /* Save old DTA */
    intdosx(&r, &r, &sr);
    OldDTASeg = sr.es;
    OldDTAOff = r.x.bx;

    r.h.ah = 0x1a;          /* Set TSR's DTA */
    sr.ds = TsrDTASeg;
    r.x.dx = TsrDTAOff;
    intdosx(&r, &r, &sr);

    r.h.ah = 0x51;
    intdos(&r, &r);         /* Save old PSP */
    OldPSP = r.x.bx;

    r.h.ah = 0x50;          /* Set TSR's PSP */
    r.x.bx = TsrPSP;
    intdos(&r, &r):

    sound(1200); delay(100); sound(1500); delay(100); nosound();
    buf = (char *) malloc(Imagesize(0,0,216,9));
    Getimage(0,0,216,9,buf,ActivePage);
    RectXor(x, y, deltax, deltay);
    while(!quit) {
        key = bioskey(0);
        switch(key) {
```

```
case ESC   : quit = TRUE;
             break;
case UP    : if(IncDecFlag==POST)
             {
             y -= d;
             y = (y<0) ? 0 : y;
             }
             else
             {
             deltay -= d;
             deltay = (deltay<1) ? 1 : deltay;
             }
             break;
case DOWN : if(IncDecFlag==POST)
             {
             y += d;
             y = ((y+deltay)>=MaxY) ? y-d : y;
             }
           else
             {
             deltay += d;
             deltay = ((deltay+y)>=MaxY) ? deltay-d : deltay;
             }
             break;
case RIGHT: if(IncDecFlag==POST)
             {
             x += d;
             x = ((x+deltax)>=MaxX) ? x-d : x;
             }
             else
             {
             deltax += d;
             deltax = ((deltax+x)>=MaxX) ? deltax-d : deltax;
             }
             break;
case LEFT : if(IncDecFlag==POST)
             {
             x -= d;
             x = (x<0) ? 0 : x;
             }
             else
             {
             deltax -= d;
             deltax = (deltax<1) ? 1 : deltax;
```

```
                    }
                    break;
case SPACE: IncDecFlag = (IncDecFlag==POST) ? DELTA : POST;
                    break;
case PLUS : d = (++d>50) ? 50 : d;
                    break;
case MINUS: d = (--d<1) ? 1 : d;
                    break;
case Alt_I: if(DispInfo==ON)
                    {
                    Putimage(0,0,buf,ActivePage);
                    DispInfo=OFF;
                    }
                    else
                    DispInfo=ON;
                    break;
case Alt_K: RectXor(x, y, deltax, deltay);
                    Dark(x, y, deltax, deltay);
                    RectXor(x, y, deltax, deltay);
                    break;
case Alt_T: RectXor(x, y, deltax, deltay);
                    Light(x, y, deltax, deltay);
                    RectXor(x, y, deltax, deltay);
                    break;
case Alt_R: RectXor(x, y, deltax, deltay);
                    Reverse(x, y, deltax, deltay);
                    RectXor(x, y, deltax, deltay);
                    break;
case Alt_G: free(grpdata);
                    grpdata = (char *) malloc
                              (Imagesize(x, y, x+deltax, y+deltay));
                    RectXor(x, y, deltax, deltay);
                    Getimage(x, y, x+deltax, y+deltay, grpdata, ActivePage);
                    RectXor(x, y, deltax, deltay);
                    break;
case Alt_D: RectXor(x, y, deltax, deltay);
                    Putimage(x, y, grpdata, ActivePage);
                    RectXor(x, y, deltax, deltay);
                    break;
case Alt_S: {
                    char filename[13];

                    Getfilename(filename);
                    RectXor(x, y, deltax, deltay);
```

```
                    Saveimage(filename, x, y, deltax, deltay);
                    RectXor(x, y, deltax, deltay);
                    break;
                    }
        case Alt_L: {
                    char filename[13];

                    Getfilename(filename);
                    RectXor(x, y, deltax, deltay);
                    Loadimage(filename, x, y);
                    RectXor(x, y, deltax, deltay);
                    break;
                    }
        case Alt_F: RectXor(x, y, deltax, deltay);
                    Flipflop(x, y, deltax, deltay);
                    RectXor(x, y, deltax, deltay);
                    break;
        case Alt_M: RectXor(x, y, deltax, deltay);
                    Mirror(x, y, deltax, deltay);
                    RectXor(x, y, deltax, deltay);
                    break;
        case Alt_P: RectXor(x, y, deltax, deltay);
                    Hercules_printscr(x, y, x+deltax, y+deltay, ActivePage);
                    RectXor(x, y, deltax, deltay);
                    break;
      }
      RectXor(ox, oy, odeltax, odeltay);
      RectXor(x,   y,  deltax,  deltay);
      ox = x;
      oy = y;
      odeltax = deltax;
      odeltay = deltay;
      if(DispInfo == ON)
      {
        char string[30];

        sprintf(string, "X=%3d Y=%3d WX=%3d Wy=%3d", x, y, deltax, deltay);
        OutBitmapText(0,0,string,NORMAL);
      }
  }
      RectXor(x,   y,  deltax,  deltay);
      Putimage(0,0,buf,ActivePage);
      free(buf);
```

```
    sr.ds  = OldDTASeg;    /* Restore Old DTA */
    r.x.dx = OldDTAOff;
    r.h.ah = 0x1a;
    intdosx(&r, &r, &sr);

    r.h.ah = 0x50;         /* Restore Old PSP */
    r.x.bx = OldPSP;
    intdos(&r, &r);
}

void RectXor(int sx, int sy, int dx, int dy)
{
    HLineXor(sy, sx, sx+dx, ActivePage);
    VLineXor(sx+dx, sy+1, sy+dy, ActivePage);
    HLineXor(sy+dy, sx+dx-1, sx, ActivePage);
    VLineXor(sx, sy+dy-1, sy+1, ActivePage);
}

void OutBitmapText(int x, int y, char *string, int mode)
{
    register int i, j;
    int len;

    HerculesMapping();
    len = strlen(string);
    for(i=0; i<len; i++)
        for(j=0; j<8; j++)
          if(mode==NORMAL)
          pokeb((ActivePage) ? 0xb800 : 0xb000, address[j+y]+i+x,
            peekb(0xf000, 0xfa6e+string[i] * 8+j));
       else
       pokeb((ActivePage) ? 0xb800 : 0xb000, address[j+y]+i+x,
            peekb(0xf000, 0xfa6e+string[i] * 8+j));
}

void Reverse(int sx, int sy, int dx, int dy)
{
    int x, y, ex, ey;

    ex = sx + dx;
    ey = sy + dy;
    for(x=sx; x<=ex; x++)
      for(y=sy; y<=ey; y++)
      if(Getpixel(x, y, ActivePage))
```

```
            Putpixel(x, y, 0, ActivePage);
          else
            Putpixel(x, y, 1, ActivePage);
  }

  void Light(int sx, int sy, int dx, int dy)
  {
      register int i;

      for(i=sx; i<=sx+dx; i+=2)
        VLine(i, sy, sy+dy, 1, ActivePage);
  }

  void Dark(int sx, int sy, int dx, int dy)
  {
        register int i;

        for(i=sx; i<=sx+dx; i+=2)
        VLine(i, sy, sy+dy, 0, ActivePage);
  }

  void Getfilename(char *filename)
  {
      char *bkdata;
      int  quit=FALSE;
      int  key;
      int  CharCount=0;
      char ch;

      delay(0);
      filename[0]='\0';
      bkdata = (char *) malloc(Imagesize(0,0,160,9));
      Getimage(0,0,160,9,bkdata,ActivePage);
      OutBitmapText(0,0,"Filename:              ",REVERSE);
      while(!quit) {
      if(kbhit()) key = bioskey(0);
      else        key = 0;
      switch(key) {
        case BACK : if((--CharCount)<0)
                    {
                    sound(1000); delay(100); nosound();
                    CharCount++;
                    break;
                    }
```

```
                    filename[CharCount] = '\0';
                    break;
        case ENTER: quit = TRUE;
                    break;
        case SPACE: break;
        default   : ch = (char) (key & 0xff);
                    if(ch==0) break;
                    if((++CharCount)>8)
                    {
                    sound(1000); delay(100); nosound();
                    CharCount--;
                    break;
                    }
                    filename[CharCount-1] = ch;
                    filename[CharCount] = '\0';
                    OutBitmapText(9,0,"          ",REVERSE);
                    OutBitmapText(9,0,filename,REVERSE);
                    break;
    }
    if(key!=ENTER) {
      OutBitmapText(CharCount+9, 0, "_",REVERSE);
      OutBitmapText(CharCount+9, 0, " ",REVERSE);
    }
}
    strcat(filename, ".TPC");
    Putimage(0,0,bkdata,ActivePage);
    free(bkdata);
}

void Saveimage(char *filename, int sx, int sy, int dx, int dy)
{
    char *buf;
    FILE *fp;
    unsigned size;

    size = Imagesize(sx, sy, sx+dx, sy+dy);
    buf = (char *) malloc(size);
    Getimage(sx, sy, sx+dx, sy+dy, buf, ActivePage);
    if((fp=fopen(filename, "wb"))==NULL) {
         char *bk;
         bk = (char *) malloc(Imagesize(0,0,160,9));
         Getimage(0,0,160,9,bk,ActivePage);
         sound(1000); delay(100); nosound();
         OutBitmapText(0,0,"File error writing!",REVERSE);
```

```
            getch();
            Putimage(0,0,bk,ActivePage);
            free(bk);
            return;
        }
        fwrite(buf, size, 1, fp);
        fclose(fp);
        free(buf);
}

void Loadimage(char * filename, int sx, int sy)
{
    char * buf;
    FILE * fp;
    unsigned size;

if((fp=fopen(filename, "rb"))==NULL) {
        char * bk;
        bk = (char * ) malloc(Imagesize(0,0,160,9));
        Getimage(0,0,160,9,bk,ActivePage);
        sound(1000); delay(100); nosound();
        OutBitmapText(0,0,"File error reading!",REVERSE);
        getch();
        Putimage(0,0,bk,ActivePage);
        free(bk);
        return;
}
    size = filelength(fileno(fp));
    buf = (char * ) malloc(size);
    fread(buf, size, 1, fp);
    Putimage(sx, sy, buf, ActivePage);
    fclose(fp);
    free(buf);
}

void Flipflop(int sx, int sy, int dx, int dy)
{
    int ex, ey;
    int i, j;
    unsigned color;
    char * temp;

    ex = sx + dx;
    ey = sy + dy;
```

```
        for(i=sy; i<=ey; i++)
              for(j=sx; j<=ex; j++)     {
              color = Getpixel(j, i, ActivePage);
              Putpixel(j, ey-i+sy, color, abs(1-ActivePage));
          }
        temp = (char *) malloc(Imagesize(sx, sy, ex, ey));
        Getimage(sx, sy, ex, ey, temp, abs(1-ActivePage));
        Putimage(sx, sy, temp, ActivePage);
        free(temp);
  }

  void Mirror(int sx, int sy, int dx, int dy)
  {
          int ex, ey;
          int i, j;
          unsigned color;
          char * temp;

          ex = sx + dx;
          ey = sy + dy;
          for(i=sy; i<=ey; i++)
          for(j=sx; j<=ex; j++)     {
                color = Getpixel(j, i, ActivePage);
                Putpixel(ex-j+sx, i, color, abs(1-ActivePage));
          }
          temp = (char *) malloc(Imagesize(sx, sy, ex, ey));
          Getimage(sx, sy, ex, ey, temp, abs(1-ActivePage));
          Putimage(sx, sy, temp, ActivePage);
          free(temp);
}

int Getpixel(unsigned x, unsigned y, int page)
{
      unsigned bitsno;
      unsigned char data;

      bitsno  = 7-(x%8);
      data = peekb((page) ? 0xb800 : 0xb000, address[y] + x/8);
      return( ( data & (1<<bitsno) ) >> bitsno );
}

void Putpixel(unsigned x, unsigned y, int color, int page)
{
   unsigned bitsno;
```

```
    unsigned char data;

    bitsno  = 7-(x%8);
    data = peekb((page) ? 0xb800 : 0xb000, address[y] + x / 8);
    if(color)
      pokeb((page) ? 0xb800 : 0xb000, address[y] + x / 8, data    (1<<bitsno));
    else
      pokeb((page) ? 0xb800 : 0xb000, address[y] + x / 8, data & (~(1<<bitsno)));
}

unsigned Imagesize(int sx, int sy, int ex, int ey)
{
      unsigned dx, dy;

      dx = (ex - sx + 1)/8 + 1;
      dy = ey - sy + 1;
      return((unsigned) dx * dy + 6);
}

void Getimage(int sx, int sy, int ex, int ey, char *buf, int page)
{
    int x, y;
    int bits;

    HerculesMapping();
    *(buf++) = (ex - sx) % 256;
    *(buf++) = (ex - sx) / 256;
    *(buf++) = (ey - sy) % 256;
    *(buf++) = (ey - sy) / 256;
    *buf = 0;
    bits = 7;
    for(y=sy; y<=ey; y++)  {
      for(x=sx; x<=ex; x++)    {
            if(Getpixel(x, y, page))
                  *buf |= 1 << bits;
            if(--bits<0 )  {
                  bits = 7;
                  *(++buf) = 0;
            }
      }
      *(++buf) = 0;
      bits = 7;
}
}
```

```
void Putimage(int sx, int sy, char *buf, int page)
{
      int x, y, ex, ey;
      int bits;

      HerculesMapping();
      ex = ((unsigned)buf[1] & 0xff) * 256 + ((unsigned) buf[0] & 0xff) + sx;
      ey = ((unsigned)buf[3] & 0xff) * 256 + ((unsigned) buf[2] & 0xff) + sy;
      buf+=4;
      bits = 7;

      for(y=sy; y<=ey; y++)      {
       for(x=sx; x<=ex; x++)     {
              if(((*buf)>>bits)&1) Putpixel(x, y, 1, page);
              else                 Putpixel(x, y, 0, page);
              if(--bits<0)      {
                     bits = 7;
                     buf++;
              }
       }
       bits = 7;
       buf++;
}
}

void VLine(int x, int sy, int ey, int color, int page)
{
     int y;

     if(sy>ey) { y=sy; sy=ey; ey=y; }
     for(y=sy; y<=ey; y++)
       Putpixel(x, y, color, page);
}

void HLine(int y, int sx, int ex, int color, int page)
{
     int x;

     if(sx>ex) { x=sx; sx=ex; ex=x; }
     for(x=sx; x<=ex; x++)
       Putpixel(x, y, color, page);
}
```

```
void VLineXor(int x, int sy, int ey, int page)
{
      int y;

      if(sy > ey) { y=sy; sy=ey; ey=y; }
      for(y=sy; y< =ey; y++)
       if(Getpixel(x, y, page))
            Putpixel(x, y, 0, page);
       else
            Putpixel(x, y, 1, page);
}

void HLineXor(int y, int sx, int ex, int page)
{
      int x;

      if(sx > ex) { x=sx; sx=ex; ex=x; }
      for(x=sx; x< =ex; x++)
       if(Getpixel(x, y, page))
              Putpixel(x, y, 0, page);
       else
            Putpixel(x, y, 1, page);
}

void p_putc(char ch)
{
       r.h.ah = 0;
       r.h.al = ch;
       r.x.dx = 0;
       int86(0x17, &r, &r);
}

void Hercules_printscr(int sx, int sy, int ex, int ey, int page)
{
      int deltax;
      int n1, n2;
      int i, j, k;
      char data;

      HerculesMapping();
      p_putc(ESC);
      p_putc('3');
      p_putc(24);
      deltax = ex - sx + 1;
```

```
        n1 = deltax % 256;
        n2 = deltax / 256;
        for(i=sy; i<=ey; i+=8)  {
         p_putc(ESC);
         p_putc('L');
         p_putc(n1);
         p_putc(n2);
         for(j=sx; j<=ex; j++)        {
                data = 0;
                for(k=0; k<8; k++)
                if((i+k)<=ey)
                    if(Getpixel(j, i+k, page))
                      data |= (1<<(7-k));
                p_putc(data);
         }
         p_putc('\n');
         p_putc('\r');
}
    p_putc(ESC);
    p_putc('2');
}

void HerculesMapping()
{
    int i;
    for(i=0; i<348; i++)
      address[i] = 8192 * (i%4) + 90 * (i/4);
}
```

7.9 结束语

许多人不喜欢以高级语言来编写驻留程序，其原因除了所产生的运行码不够简洁外，不能完全了解大部分函数内部的操作情形也是原因之一。所幸 C 语言除了具备高级语言的特色外，更具备了强大的低级处理能力，使我们在写驻留程序方面不至于缚手缚脚。同时 Turbo C 为了让用户更清楚的了解程序的起始及结束到底做了哪些处理，亦提供其源代码 C0.ASM 供用户参考。如果你在写驻留程序时发现某些地方一直未能如您预期般地操作，查一查这个文件也许会有所收获。

本书限于篇幅，许多编写驻留程序的技巧、实例以及更强大的嵌入式功能都无法收录在本书之内，希望有机会能以专书的形式再加以深入探究。7.8 节所附的绘图小精灵程序仍有许多考虑不够周到之处，留待读者自行发挥，利用这个程序开发出一套优于市售绘图软件的程序，并非不可能的事。

第八章　扫描仪应用程序

8.1　扫描仪简介

图像扫描仪（Image Scanner）的功用是将图片、照片经过扫描操作，转换成计算机可以存储、处理的数据。扫描仪问市不过几年，由于市场的强烈需求，促使厂商积极研究开发，其软件的功能愈来愈强大，价格也日益普及。目前扫描仪已经在计算机的输入设备中，扮演不可缺少的角色。

扫描仪的主要应用有图形数据库、计算机绘图、动画设计、桌上排版系统、计算机辅助设计和简报系统等等，而其它的相关软件，也逐渐提供了支持扫描仪的功能。

若以输入数据的色彩区分，扫描仪有单色、灰度、彩色三种。黑白图像只有 0 与 1 两种状态，因此每个图点占用 1 比特的内存。灰阶扫描仪每个图点需要 6 或 8 位内存，故可以在屏幕上表现出 64 色或者 256 色灰阶，看起来和黑白照片十分相似。彩色扫描仪和前面两种扫描仪的构造大致相同。只不过增加了分色光或滤光镜片，将原来的彩色原稿分成红、蓝、绿三色光处理。每个彩色图点一般以 24 位存储，则最多可以呈现 16，777，266 种色彩，而目前 PC 上的软件最多仅能在屏幕上呈现 256 色。

扫描仪若以构造或者动作方式区分，又有平台式、滚轴式和掌上型三种。平台式的外形类似复印机，扫描原稿的厚度不受限制。滚轴式以滚轴输送原稿，因此纸张的长度不受限制，然而却仅限于扫描仪单张原稿。此外，如果配合自动送纸器作业，则可以做到多张原稿连续扫描。掌上型扫描仪的价格便宜，然而扫描宽度仅有 10.5 公分，且在扫描时容易因倾斜或者移动太快而影响输入质量。一般而言，平台式、滚轴式扫描仪常用于专业用途，而掌上型扫描仪较适合个人使用。

扫描仪的种类很多，在购买时，厂商通常会附赠绘图软件以供操作扫描仪之用；例如 A4Scan 的变影 72，GeniScan 的精灵龙，全友的 Eyestar，NTScan 的达文西等等。通过这些软件，用户可以扫描图形，并且加以修改、存储。如果我们希望不经过前述的绘图程序，而在自己所设计的程序下进行图形扫描，该如何着手呢？在本章中，我们以国内常见的 A4 Scanner 为例，和读者交换扫描仪应用程序的设计心得。

8.2　扫描仪应用程序设计实例

目前扫描仪的规格尚未统一，每一种厂牌的扫描仪都有各自的接口和应用软件。如果我们想自行撰写应用程序，则必须先行取得有关的技术数据。这些数据除了少数几家公司之外，一般厂商并未提供给用户。本章仅就 A4Scan 为例，介绍扫描仪应用程序的设计方法。

A4Scan 是五百科技公司所发行的单色掌上型扫描仪，它通常与变量 72 绘图软件一并出售。该公司还提供了 A4Scan 的驱动程序 A4SCAN.COM，如果我们想将 A4 Scan

应用在自己设计的程序上，不管使用何种程序设计语言，都可以通过 A4 Scan 让扫描仪动作。图 8-1 便是作者采用 A4Scan 所设计的图像处理应用软件，然而限于篇幅，在本章仅针对扫描仪的应用部分，做一简单的介绍。

图 8-1　一个支持 A4 SCAN 的应用软件

A4SCAN.COM 是一个驻留程序，它是用中断的方式来控制扫描仪任务，它的中断号码是 0A4H，以下便是 A4 Scan 所提供的各项功能：(数据来源：五百科技公司)

AX＝0　设置屏幕分辨率

输入参量

AX＝0

BX＝0	640×350标准EGA卡分辨率。
1	640×480彩色EGA中文分辨率。
2	640×480标准VGA卡分辨率。
3	640×480仅供倚天2.0版（台湾）的ET16V-16使用。
4	640×480单色中文分辨率。
5	720×348标准单色绘图卡分辨率。

系统 图样 书法 字体 区块
Motage
蒙太奇 影像处理系统
Motage Vol. 2.0
1991/2/12 莊粤盛
蒙太奇
讯息栏
OCR
END

应用在自己设计的程序上，不管使用何种程序设计语言，都可以通过 A4 Scan 让扫描仪动作。图 8-1 便是作者采用 A4Scan 所设计的图像处理应用软件，然而限于篇幅，在本章仅针对扫描仪的应用部分，做一简单的介绍。

图 8-1 一个支持 A4 SCAN 的应用软件

A4SCAN.COM 是一个驻留程序，它是用中断的方式来控制扫描仪任务，它的中断号码是 0A4H，以下便是 A4 Scan 所提供的各项功能：(数据来源：五百科技公司)

AX＝0 设置屏幕分辨率

输入参量

AX＝0

BX＝0 640×350标准EGA卡分辨率.

1 640×480彩色EGA中文分辨率.

2 640×480标准VGA卡分辨率.

3 640×480仅供倚天2.0版（台湾）的ET16V-16使用.

4 640×480单色中文分辨率.

5 720×348标准单色绘图卡分辨率.

AX＝1　设置扫描窗口

输入参量:

AX－1

CX＝窗口左上角的水平坐标（必须为 8 的倍数).

DX＝窗口左上角的垂直坐标（最小为 0，不可超过屏幕的分辨率).

DI＝窗口的水平宽度（必须为 8 的倍数).

SI＝窗口的垂直扫描线数，DI+SI 必须小于屏幕的最高分辨率.

AX＝2 设置扫描仪的硬件配置

输入参量

AX＝2

BX＝1　设置DMA 1

3　　　设置DMA 3

CX＝扫描宽度，其值由 4mm～104mm.

DX＝1,2,3,4 设置所要使用的 Io port，内定值为 1.

SI＝扫描缓冲区的开始地址.

DI＝所要使用的缓冲区大小，以 64K 的倍数表示，例如 1 表示 64K，2 表示 128K.

AX＝3 开始扫描

输入参量

AX＝3

返回参量

AX＝0 扫描成功

8 扫描仪未接到接口上.

80H 找不到扫描仪的接口.

若 AX 返回值为 0，则:

DI＝水平的点数.

SI＝垂直的点数.

BX＝数据存放起点的地址.

现在，我们以实例说明各种功能调用的使用．首先设置屏幕分辨率，最常用的是 720×348 单色绘图模式，其设置情形为:

```
r.x.ax = 0;
r.x.bx = 5;
int86(0xa4, &r, &r);
```

接着设置扫描窗口，如果我们让扫描输入的图形显示于屏幕左上角(8,8)，右下角(632,320) 的区块内，则程序如下:

```
r.x.ax = 1;
r.x.bx = 8;
```

```
r.x.dx=8;
r.x.di=632;
r.x.si=320;
int86(0xa4, &r, &r);
```

在开始扫描之前，扫描仪有关的硬件配置应加以设置，这项任务通过调用功能 2 完成。假使我们希望使用 DMA1，扫描宽度为 104mm，存储扫描图形的缓冲区以 8000H 为起点，共占 128KB，则硬件组态的设置如下：

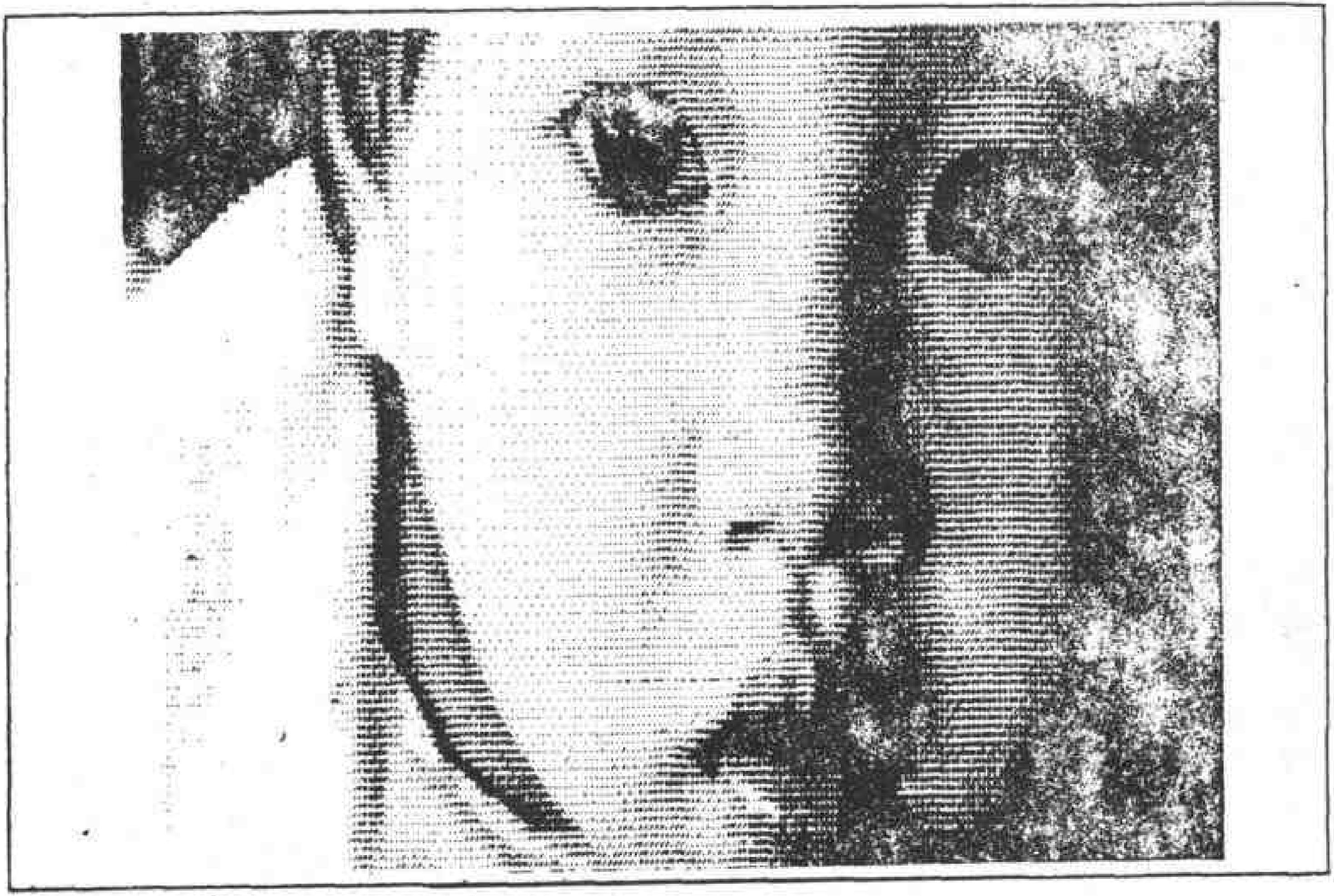
```
r.x.ax=2;
r.x.bx=1;
r.x.xc=104;
r.x.cx=2;
r.x.si=0x8000;
r.x.di=2;
int86  (0xa4, &r, &r);
```

最后，让扫描仪开始操作，我们只要输入下列语句即可：

```
r.x.ax=3;
int86(0xa4,&r,&r);
```

图 8-2　SCAN1.C 程序运行结果

程序 SCAN1.C 就是根据前述的例子写成的，它能驱动扫描操作。在着手扫描之前，必须先将 A4 扫描仪安装妥当，接着载入五百科技公司的扫描仪驱动程序：



```
r.x.dx=8;
r.x.di=632;
r.x.si=320;
int86(0xa4, &r, &r);
```

在开始扫描之前，扫描仪有关的硬件配置应加以设置，这项任务通过调用功能 2 完成。假使我们希望使用 DMA1，扫描宽度为 104mm，存储扫描图形的缓冲区以 8000H 为起点，共占 128KB，则硬件组态的设置如下：

```
r.x.ax=2;
r.x.bx=1;
r.x.xc=104;
r.x.cx=2;
r.x.si=0x8000;
r.x.di=2;
int86  (0xa4, &r, &r);
```

最后，让扫描仪开始操作，我们只要输入下列语句即可：

```
r.x.ax=3;
int86(0xa4,&r,&r);
```

图 8-2　SCAN1.C 程序运行结果

程序 SCAN1.C 就是根据前述的例子写成的，它能驱动扫描操作。在着手扫描之前，必须先将 A4 扫描仪安装妥当，接着载入五百科技公司的扫描仪驱动程序：

C>A4SCAN　　<ENTER>

如果安装成功，屏幕会出现下列讯息：

Copyright(c) 1990, A4THCH Co..LTD.

A4 Scan driver installed ok!

此时便可以运行我们的图形扫描程序了：

C>SCAN　　<ENTER>

假使一切正常，则 A4 扫描仪便会发出黄色光等待扫描。当我们按下 Start 键开始扫描图形，屏幕便向下滚出输入的图像了。图 8-2 便是 SCAN1.C 程序的运行实例。

```
/* SCAN1.C */

#include "dos.h"
#include "graphics.h"
main()
{
  int driver=DETECT,mode=0;
  union REGS r;
  initgraph(&driver,&mode,"");;
   pokeb(0x40,0x49,5);
   setvisualpage(1);
   setactivepage(1);

   r.x.ax=0;
   r.x.bx=5;
   int86(0xa4,&r,&r);

   r.x.ax=1;
   r.x.cx=8;
   r.x.dx=8;
   r.x.di=632;
   r.x.si=320;
   int86(0xa4,&r,&r);

   r.x.ax=2;
   r.x.bx=1;
   r.x.cx=104;
   r.x.dx=2;
   r.x.si=0x8000;
   r.x.di=1;
   int86(0xa4,&r,&r);

   r.x.ax=3;
```

```
34:    int86(0xa4,&r,&r);
35:
36:    closegraph();
37:}
```

程序说明:

9—12　进入绘图模式，并设置显示页为第1页。

14—16　设置屏幕分辨率。

18—23　设置扫描窗口。

25—31　设置扫描仪的硬件配置。

33—34　开始扫描。

36　结束绘图模式。

接着，我们将 SCAN1.C 程序的功能加以扩充，使得在运行扫描时，屏幕先出现扫描窗口，而在扫描结束之后，能够由键盘输入文件名称，并且以这个文件名称将图形存入磁盘。SCANNER.C 便是这个扩充功能的程序，而它的运行结果则如图 8-3 所示。

```
/ * SCANNER.C * /

#include "dos.h"
#include "stdio.h"
#include "graphics.h"
main()
{
    int driver=DETECT,mode=0;
    initgraph(&driver,&mode,"");
     pokeb(0x40,0x49,5);
     setvisualpage(1);
     setactivepage(1);
     main__screen();
     scan1();
     save__image();
     closegraph();
}

  main__screen()
{
  FILE  * fp;
  char  * buff;
  long size;
      buff=(char * )malloc(1655);
      fp=fopen("scantit.blk","rb");
      fread(buff,1,1655,fp);
      fclose(fp);
      rectangle(0,0,719,347);
```

```
        rectangle(16,40,700,338);
        setfillstyle(11,0);
        floodfill(3,3,1);
        putimage(210,2,buff,0);
        free(buff);
}

scan1()
{
 union REGS r;
      r.x.ax=0;
      r.x.bx=5;
      int86(0xa4,&r,&r);

      r.x.ax=1;
      r.x.cx=32;
      r.x.dx=48;
      r.x.di=656;
      r.x.si=288;
      int86(0xa4,&r,&r);

      r.x.ax=2;
      r.x.bx=1;
      r.x.cx=104;
      r.x.dx=2;
      r.x.si=0x8000;
      r.x.di=1;
      int86(0xa4,&r,&r);

      r.x.ax=3;
      int86(0xa4,&r,&r);
}

save_image()
{
int size;
FILE *fp;
char *buff,message[1600],*filename;
      size=imagesize(32,48,656,288);
      buff=(char *)malloc(size);
      getimage(32,48,656,288,buff);
      fp=fopen("scanmsg.blk","rb");
      fread( message,1,1522,fp);
      putimage(250,150,message,0);
```

```
73:        fclose(fp);
74:
75:        input__filename( filename);
76:        fp=fopen(filename,"wb");
77:        fwrite(buff,1,size,fp);
78:        fclose(fp);
79:        free(buff);
80:}
81:
82:input__filename( filename )
8 3:char  * filename;
8 4:{
85:     int i=0;
86:     char ch;
87:
88:      setviewport(341,177,448,188,1);
89:     clearviewport();
90:     clearviewport();
91:     do  {
92:            ch=getch();
93:            if(ch= =0x0d)         {
94:                    strcat( filename,0 );
95:                    break;
96:            };
97:            if( ch= =0x08 )             {
98:                    i--;
99:                     * ( filename+i )=0;
100:                     clearviewport();
101:                     outtextxy( 1,3,filename );
102:            }
103:            else              {
104                     * ( filename+i )=ch;
105:                      * ( filename+i+1 )=0;
106:                     outtextxy( 1,3,filename );
107:                     i++;
108:            }
109:} while( ch != 0x0d );
110:}
```

程序说明:

6-16　　主程序。进入绘图模式，并调用主画面、扫描图形、存储图像文件等函数。

19-33　　主画面函数，以 rectangle及 floodfill等函数绘制窗口，并以 fread、putimage 等函数将预存的标题显示在屏幕上。

36–59　　扫描函数.

61–80　　存储扫描的图形，其文件名称则由input__filename函数自键盘取得.

82–110　　以交互方式自键盘输入图形文件的名称.

(1)扫描图形

扫描器应用程序

36–59　　扫描函数.
61–80　　存储扫描的图形，其文件名称则由input__filename函数自键盘取得.
82–110　　以交互方式自键盘输入图形文件的名称.

(1)扫描图形

(2)输入要存储的图形文件名称

图 8-3 SCANNER.C 程序的运行结果

8.3 结束语

一般而言，不同厂牌的扫描仪在它本身的硬件结构上大致相同，而其接口卡和驱动程序的差异较大，彼此不能互通使用。本章所介绍的 A4 Scan 扫描仪在国内很常见，读者如要给其它厂牌的扫描仪设计应用程序，请自行参阅厂商所提供的技术数据。

限于篇幅，我们仅将扫描仪做简单的介绍，读者如有兴趣，可将本章程序加和图形绘制、编辑、文字输入等功能，那么您也可以自行编写像达文西、变影 72 那样的绘图软件了。

扫描器应用程序
存储
请输入文件名:
TEST1.BLK

(2)输入要存储的图形文件名称

图 8-3　SCANNER.C 程序的运行结果

8.3　结束语

一般而言，不同厂牌的扫描仪在它本身的硬件结构上大致相同，而其接口卡和驱动程序的差异较大，彼此不能互通使用．本章所介绍的 A4 Scan 扫描仪在国内很常见，读者如要给其它厂牌的扫描仪设计应用程序，请自行参阅厂商所提供的技术数据．

限于篇幅，我们仅将扫描仪做简单的介绍，读者如有兴趣，可将本章程序加和图形绘制、编辑、文字输入等功能，那么您也可以自行编写像达文西、变影 72 那样的绘图软件了．

Document generated by Anna’s Archive around 2023-2024 as part of the DuXiu collection (https://annas-blog.org/duxiu-exclusive.html).

Images have been losslessly embedded. Information about the original file can be found in PDF attachments. Some stats (more in the PDF attachments):
{
  "before_pdg2pic_conversion": {
    "filename": "MTExMDU0NzMuemlw",
    "filename_decoded": "11105473.zip",
    "filesize": 5714398,
    "md5": "a609a9af8d15860734721ea70a1e4ce0",
    "header_md5": "3bcf4a4d1739b8fc019c229bf4dba7e9",
    "sha1": "787e0f6e8266fc10f89ead843a78601c9ea73150",
    "sha256": "3f27d7f42f97dea97a8c6b4ccd0a6e7527bb4d3c70ea3321a25b93b05d3cd846",
    "crc32": 4137671099,
    "zip_password": "",
    "uncompressed_size": 5961100,
    "pdg_dir_name": "",
    "pdg_main_pages_found": 351,
    "pdg_main_pages_max": 351,
    "total_pages": 358,
    "total_pixels": 13384394
  },
  "after_pdg2pic_conversion": {
    "filename": "MTExMDU0NzMuemlw",
    "filename_decoded": "11105473.zip",
    "filesize": 24100664,
    "md5": "12c8b15bdc7f04c3301e91dcfa75a8e7",
    "header_md5": "cc6bd1f52075715ea84ca51769296e40",
    "sha1": "a0fdd7ccc08d05fb4b6fd8474f97e1a53bbb224b",
    "sha256": "71bcbdabf6a884e7c404f6690dee689136c27e091f283abe698c2b711d654006",
    "crc32": 2566171,
    "zip_password": "",
    "uncompressed_size": 25233349,
    "pdg_dir_name": "",
    "pdg_main_pages_found": 351,
    "pdg_main_pages_max": 351,
    "total_pages": 358,
    "total_pixels": 593162846
  },
  "pdf_generation_missing_pages": false
}